Robot speech recognition

This is going to be a rather involved chapter, but all of the concepts are fairly easy to understand and I was very happy with the results I achieved when I developed this chapter. We will end up with a lot of collateral abilities for our robot that we are getting *for free*, and will end up with a very strong framework to build voice recognition and commands upon. Let's get right to it.

What are we doing?

We set several goals for our robot in Chapter 2, which included being able to give voice commands to the robot, since we may be using the robot without a base station. I also wanted the robot to be able to interact with my grandchildren, and specifically to be able to tell and respond to knock-knock jokes, a favorite activity of my grandson, William.

We can break this process down into several steps, which we will be handling independently. We need the robot to be able to *hear*, or have the ability to convert sound into a digital form. We need to process sounds into words, which is to say, turn sounds into text.

We need to not just recognize individual words, but to combine those words into sentences and from those sentences, infer the intent of the speaker to understand what the robot is to do. We do not want to use canned or memorized speech commands, but rather have the robot be able to do some **natural language processing** (**NLP**) to create a form of robot understanding of the spoken word.

For example, if we want to have a command for *pick up a toy*, we humans could phrase that several ways: *grab a toy*, *grasp a toy*, *pick up that toy car*, or even *get that*. We want the robot to understand or at least respond to all of those utterances with the same action, to drive to the nearest toy and pick it up with the robot arm.

The other half of the interface is that the robot needs to respond back by speaking. Text-to- speech systems are fairly commonplace today, but we would like to have some natural variations in the robot's speech patterns to help make the illusion that the robot is smarter than it really is.

Our steps for this chapter are as follows:

1. Receive audio (sound) inputs.
2. Convert those sounds into text that the robot can process.
3. Use processing on those text words to understand the intent of the speaker.
4. Use that intent as a command to perform some task.
5. Provide verbal responses in the form of spoken words (text to speech) back to the operator to confirm the robot heard and understood the command.
6. Create a custom verbal interface that both tells and responds to knock-knock jokes.

Speech to text

In the rest of this chapter, we will be implementing an AI-based voice recognition and response system in the robot and creating our own custom voice interface. We will be using Mycroft, an open source voice activated *digital assistant* that is adept at understanding speech and is easily extended for new functions and custom interfaces.

The process we will use for voice interaction with the robot follows this script:

1. **Wake word** (Hey, Albert)
2. Pause for the robot to make a **beep** sound to show it is listening
3. Command or query from human (*move forward one step*)
4. Robot responds verbally (*moving forward six inches*)

There are two forms of text-to-speech involved in this process that greatly simplify matters for the robot. First, the robot is listening continuously for only one sound – the *wake word*. This is a specific sound that just means one thing – get ready to process the next sound into a command. Why is this necessary?

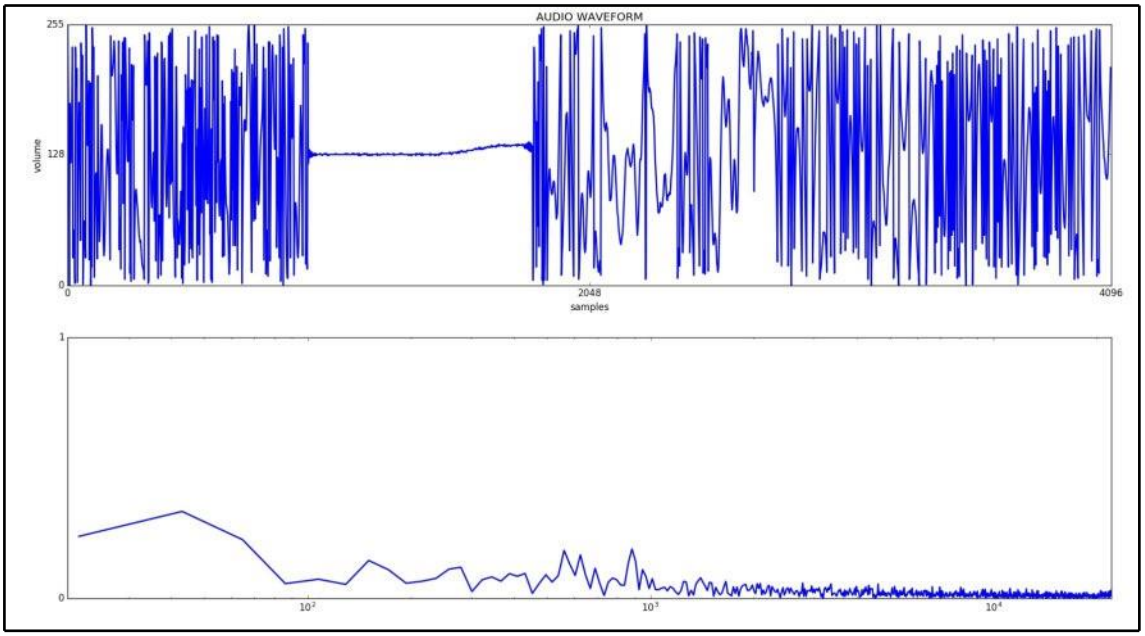
Since the robot has only a very small processor – the Raspberry Pi – it really does not have the sort of onboard compute power to run a robust speech-to-text (STT) engine. But it can run a simple sound recognizer that can listen for just one thing – the *wake word*. You are familiar with this from other voice command systems, such as Alexa or Siri, that also either use a special wake word or a button to have the interface pay attention.

Once the wake word is received, the Raspberry Pi switches into record mode, and records the next thing we say. It then transfers that information to an online system, the Google Cloud Speed to Text system (the same thing that runs the Google Assistant).

How does the robot recognize the wake word? The speech system we will be using, the open source system **Mycroft**, uses one of two methods. The first is a **phoneme recognition** system called Sphynx. What the heck is a **phoneme**? You can understand that words are made up out of individual sounds, which we roughly assign to letters of the alphabet. An example would be the *p* sound in the word *pet* or *pick*. We make the *pppp* sound by putting our lips together. The burst of sound we make is identifiable as a *P* sound – this is a phoneme. The word Albert has several phonemes – the *A* sound, (*ah*), the *L* sound, the *B*, the *ER* together (*errrrr*), and finally, the *T*. The letters we associate with the sounds – the *ch* in *cherry*, the *er* in Albert, are called graphemes, as they graphically represent these sounds. We could say that the speech-to -text problem is one of mapping these phonemes to graphemes, but we know that this is too easy – English has all sorts of borrowed words and phrases where the pronunciation and the spelling are far apart.

The frontend of the Mycroft speech recognition process uses phonemes to recognize the wake word. You will find that it is quite sensitive. I had no problem getting the speech processor to receive the wake word from eight feet away. When we get to the setup section, we will change the default Mycroft wake word from *Hey, Mycroft*, to *Hey, Albert*.

What is the other method for receiving the wake word? Mycroft can also use a trained neural network that has been taught to recognize entire words all at once by their spectral power graph. What is a spectral graph? Your voice sound is not one frequency of sound energy – it is a complex congregation of different frequencies produced by our mouths and vocal cords. If we spoke in pure frequencies, we would sound like a flute- pure tones at mostly one frequency. We can use a process called a fast **fourier** transform to convert a selection of speech into a graph that shows the amount of energy (volume) at each frequency. This is called a **spectral plot** or **spectral graph**. The low frequencies are at the left, and higher frequencies at the right. Most of human speech energy is concentrated between frequencies between 300 Hz and 4,000 Hz. Each word has a unique distribution of sound energy amounts in these frequencies, and can be recognized by a neural network in this manner:



Both the phoneme method and the neural network method use spectral plots to recognize sounds as words, but the phoneme process divides words into individual sounds, and the neural network listens and recognizes the entire word all at once. Why does this make a big difference? The phoneme system can be developed to recognize any word in English without reprogramming or retraining, while the neural network has to be trained on each word individually, and hopefully by a lot of different speakers with a lot of different accents.

Our next step after receiving the wake word is to record the next sounds that the robot hears. The Mycroft system then transfers that audio data over the internet to the Google online speech-to-text engine (`https://cloud.google.com/speech-to-text/`). This is a quick way to resolve the problem of our little Raspberry Pi not having enough processing power or storage to have a robust speech recognition capability.

What goes on the in the Google Cloud? The STT engine breaks the speech down into phonemes (sounds) and uses a neural network to assign the most probable graphemes (letters) to those sounds. The output would be spelled out more phonetically than want to receive. For example, the sentence *How many ounces in a gallon?* will come out *HH AW . M EH N IY . AW N S AH Z . IH N . AH . G AE L AH N*. (source: *CMU Pronouncing Dictionary*) How is this the case? What happened? These are the phonemes that make up that sentence. The periods indicate spaces between words. Now the system has to convert this into the words we are expecting. The STT system uses word rules and dictionaries to come up with the most likely conversion into regular words. This includes both expert systems (word rules) as well as trained neural networks that predict output words based on phonemes.
We can call this step the *language model*. Our STT outputs the sentence *How many ounces in a gallon?* and sends it back to the robot, all in less than 2 seconds.

So the robot receives the text *How many ounces in a gallon?* What do we do with it? Lets look at the sentence and break it down into its component parts, just like we did in grade school. The type of sentence is a question, as it starts with *How*. The subject of the sentence is *How Many*. The verb is *are*, which is implied in the form of the question (How many ounces **are** in a gallon). The object is *ounces* and the modifier (or adjective) is *in a gallon*.

Intent

The natural language processing we are doing has one aim, or goal. We are giving commands to our robot using a voice interface. Commands in English normally follow a sentence pattern, something like *You – do this*. Often the *you* subject of the sentence is implied or understood, and left out. We are left with statements like *Clean this room*, or *Pick up those toys*. The intent of these commands to to have the robot initiate a program that results in the robot picking up toys and putting them away. The robot and its processor have to divine or derive the intent of the user from the words that are spoken. What we want is for any reasonable sentence to have as it's meaning, *You, robot, start your pick up toys process*.

Think of how many ways we can say that command to the robot. Here are some examples: •

- Let's clean up this room
- Put away the toys
- Pick up the toys
- Pick up all the toys
- Clean up this room
- Put those away
- Put this away
- Time to clean up

What do these phrases have in common? They all imply the subject who is doing the action is the robot. There are no words like *You, robot, Tinman* to indicate to whom the command in intended. The word *toys* appears a lot, as does *pick*, *clean*, and *put away*. It is possible that we can just pay attention to those keywords to understand this command. If we get rid of all of the common conjunction and pronoun words, what does the list look like?

- Clean room

- Put toys
- Pick toys
- Pick toys
- Clean room
- Put away
- Put away
- Time clean

An important concept for this chapter is to understand that we are not trying to understand all speech, but only that subset of speech that are commands that the robot can execute.That list is fairly short. The robot can only be told to pick up toys, drive around, move its arm, and stop. That is about it.

A general solution to this voice recognition problem would be to have some ability to predict from the command given to the robot, the likelihood that intent of the user points to one command more than any of the others. You can see that in the case of the word *clean*, none of our other commands (drive around, move arm, or stop) relate to *clean* at all. Thus a sentence with *clean* in it most probably is associated with the pick up toys command.

This process of deciding intent will be used later in this chapter to send commands to the robot. We will use an open source AI engine called Mycroft to accomplish this.

Now we are going to jump right into programming the TinMan robot to listen and understand commands using a open source artificial intelligence package called Mycroft. Mycroft is a version of a digital assistant similar to Siri from Apple or Alexa from Amazon, in that it can listen to voice commands in a mostly normal fashion and interface those commands to a computer. We are using it because it has an interface that runs on a Raspberry Pi 3. Here we go.

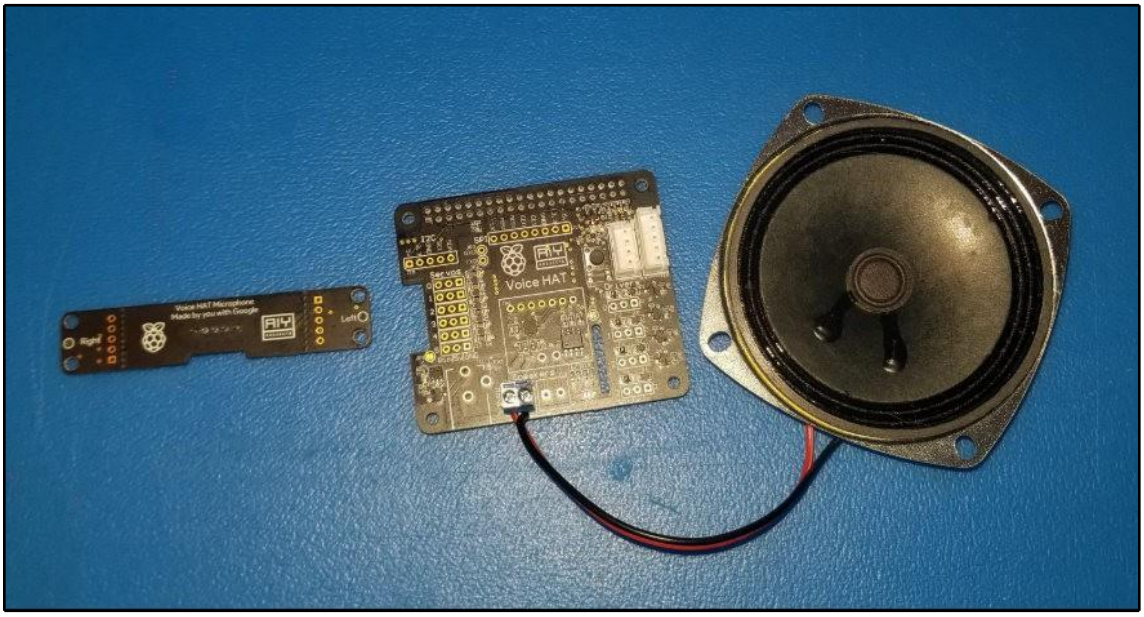
Mycroft

Installing Mycroft on Rapsberry Pi 3.

Hardware

One of the few things that the Raspberry Pi did not come with is audio capability. It has no speakers or microphones. I found a quick and effective way to add that capability was to use an existing hardware kit that provided both a very high quality speaker and an excellent set of stereo microphones in a robot-friendly form factor. Note that this works only with the Raspberry Pi 3 board – it will not work with earlier Raspberry Pis.

The kit is the Google AIY Voice Kit. The website is `http://aiyprojects.withgoogle.com`:



I decided to use the Mycroft open source voice assistant software instead, which offered much of the same functionality but in a more user-friendly arrangement. We'll cover Mycroft in the next section after we get the hardware installed.
You will not need any of the cardboard that came with the kit. Turn off the Pi, and install the Voice Hat to the top of the Raspberry Pi 3 circuit board. Be careful to line up the pins.

The next step is to connect the speakers with the screw terminals. Connect the microphones via the JST connector. This is the connector on the top that has the five wires. That ends the hardware part of the setup.

Turn on your Raspberry Pi 3 with the new speaker and microphone. Now we

can dive into the software.

We are going to get the software drivers for the Google AIY Voice Kit from the GiHhub site for the

project. We will just be using the drivers, but we have to download the whole thing.

Go to the Google AIY project Raspian GitHub site: `https://github.com/google/aiyprojects-raspbian` , and input the following code:.

```
Sudo apt-get update Cd
```

Start the Pulse Audio daemon. You may need to reboot after this step:

```
pulseaudio –D
```

Go back to the `home` directory:

```
cd
```

Now we download the AIY project source code to our RasPi:

```
git clone https://github.com/google/aiyprojects-raspbian.git voice-recognizer-raspi

cd ~/voice-recognizer-raspi
```

These scripts will install the audio drivers:

```
sudo scripts/configure-driver.sh sudo
scripts/install-alsa-config.sh
```

Now we must reboot the Pi:

```
sudo reboot
```

After rebooting, we can test the sound set up by playing some sound:

```
speaker-test –c2
```

This will play some white noise from the speakers. You can also try the following:

```
speaker-test -c2 --test=wav -w /usr/share/sounds/alsa/Front_Center.wav
```

This will say the phrase `Front-Center`. If you don't hear the sounds, try re-installing the drivers and rebooting, and also check your wires.

Mycroft software

While there are several ways to install Mycroft, we have to put Mycroft on top of the other software we have already. Since Mycroft has to get along with the ROS, and all of the artificial intelligence packages we installed, such as TensorFlow, Theano, and Keras, it is better that we use the `git clone` method to download the source code and build Mycroft on the Raspberry Pi:

```
git clone https://github.com/MycroftAI/mycroft-core.git cd
Mycroft-core

bash dev_setup.sh
```

Mycroft will create a virtual environment it needs to run. It also isolates the Mycroft package from the rest of the packages on the Raspberry PI.

In order to get the Mycroft system to work in this manner, I also had to do one more step. The

Mycroft system kept failing when I first tried to get it to run. It would quit or get stuck when I tried to start the debugger. In order to correct this problem, I had to recompile the entire system using the following steps:

```
sudo rm -R -/.virtualenvs/Mycroft cd
~/mycroft-core

./dev_setup.sh
```

Once that is done (and it took quite a while – as in several hours), you should be able to run the Mycroft system with the startup commands:

```
./start-mycroft.sh debug
```

You can start in debug mode or:

```
./start-mycroft.sh all
```

You can start in normal mode.

You will probably be using debug mode quite a bit when you are developing your speech commands.

Now test that Mycroft is working properly. When you first get Mycroft to run, it will want to be paired with your login account on the Mycroft web server. You need to set up a services account on the Mycroft website at `http://home.mycroft.ai`. Then the Raspberry Pi will give you a six-letter code to put into the website under **Devices** (on the **Hamburger** menu on the far right hand side of the website).

Once the robot is paired with the Mycroft server, it can transfer data back and forth. The wake word will start out being the default *Hey, Mycroft*. You can test that everything is working by first asking *Hey, Mycroft, what time is it?*.

Mycroft divides its capabilities into *skills* that are each controlled by a separate script. The Time skill is totally self-contained inside the Raspberry Pi. The robot should give you a voice response that is replicated on the debug console.

Next you can ask Mycroft a more advanced skill, like looking up information on the internet. Ask "Hey, Mycroft, how many ounces in a gallon?" Mycroft will use the internet to look up the answer and reply.

For the next step, you can change the wake word on the Mycroft website to something more appropriate – we did not name this robot Mycroft. We have been calling this robot *Tinman*, but you can choose to call the robot anything you want. You may find that a very short name like *Bob* is too quick to be a good wake word, so pick a name with at least two syllables. Go to the Mycroft web page (`http://home.mycroft.ai`) and log in to your account. If you have not yet created an account, now is your chance.

Click on your name in the upper right corner and select **Settings** from the menu. You can select several settings on this page, such as the type of voice you want, the units of measurement, and time and date formats. What we want to do is change the wake word from the default (*Hey, Mycroft*) to the name of our robot (*Hey, Tinman* or *Hey, Albert*). Select the small text **Advanced Settings** in the third paragraph of the page. This will take you to the page where we can change the wake word.

We change the first field **Wake word** to **Custom**. We change the next line to put in our custom

wake word – *Hey, albert*. We also need to look up the phonemes for this wake word. Click on the **this tool** hyperlink to be taken to the **CMU Pronouncing Dictionary** at Carnegie Mellon University. Put in our phrase and you will get out the phoneme phrase **HH EY . AE L B ER T** . The phoneme syntax puts periods to show the spaces between words. Copy and paste this phrase and go back to the Mycroft page to paste the phoneme phrase into the **Phonemes** field. You are done – don't change any of the other settings. Hit **Save** at the top of the page before you navigate away.

You can test your new wake word back on the Rapsberry Pi. Start Mycroft up again in debug mode and wait for it to come up. Say your new wake phrase and enjoy the response. I have a standard test set of phrases to show Mycroft's skill at being the voice of our robot. Try the following:

- *Hey, Albert. What time is it?*
- *Hey, Albert. What is the weather for tomorrow?*
- *Hey, Albert. How many ounces in a gallon?*
- *Hey, Albert. Who is the queen of England?*

You should get the appropriate answers to these questions. Mycroft has many other skills that we can take advantage of, such as setting a timer, setting an alarm at a clock time, listening to music on Pandora, or playing the news.

What we will be doing next is adding to these skills by creating our own that are specific to our room-cleaning robot. Then we can do the knock-knock jokes.

Skills

The first skill we will create is a command to pick up toys. We are going to connect this command to the ROS to control the robot.

Dialogs

Our first step is to design our dialog on how we will talk to the robot. Start by making a list of what ways you might tell the robot to pick up the toys in the playroom. Here is my list:

- Let's clean up this room
- Put away the toys
- Pick up the toys
- Pick up all the toys
- Clean up this room
- Put those away
- Put this away
- Time to clean up
- Who made this mess?

You will note that there are several key words that are specific to the command to clean up the room. We have the word *clean*, of course. We have the phrase *pick up*, and *away*. We also have the words *toys* or *toy*, and finally *mess*. These key words will cue in the natural language processor, and allow some variation in the exact words used.

Next, we write down what we want the robot to say back. We don't want the same canned response each time; it would be good to have some natural variation in the robot's responses. Here is my list of responses, with a variety of robot attitudes represented:

- Command received – picking up toys.
- I am glad to hear the grandchildren came for a visit. I will pick up now.
- Picking up toys.
- Message understood – toy pickup commencing
- Toy pick up mode initiated.
- Time to pick up toys.
- Mary Poppins does not live here, so do not start singing
- This is my job – picking up toys is what I do.
- Toy pick operation start. Scanning for toys. Wow, there are a lot of toys!
- Toys again? OK, I have got this.

You can use as many of these as you want. The Mycroft system will grab a random phrase from this list. This gives us some room for creativity, and gives the illusion that the robot is more intelligent than it really is. This type of response system does work quickly for us to develop our dialogs.

We have to now create a skill, and fit it into the standard skills framework Mycroft uses. We will have to create a GitHub repository to put our skill into, and use Python to create a programming framework around the skill. Make sure you have a GitHub repository to put your skill into. Create a GitHub account if necessary (it's free). We start by forking the Mycroft Skill repository into our GitHub account.

Go to the Mycroft skills GitHub web page at `https://github.com/MycroftAI/mycroft-skills/`.

At the upper right, you will see three buttons: **Watch**, **Star**, and **Fork**. Hit the **Fork** button to create a copy of all the skills in your repository. We are going to use the skill template to make a new skill. Now we need to clone this repository so we can edit it. Log onto your Pi (or your development machine) and clone your repository. For me, it looked like this:

```
git clone https://github.com/FGovers/mycroft-skills-1
```

We need to create a new skill set of directories. This has to follow a specific pattern in order to work. We are going to copy the skill template (`00__skill_template`) to do this.

```
cp –R 00_skill_template skill__pickup_toys
```

What we end up with looks like this:

```
ls -l

 4 drwxrwxr-x 3 ubuntu ubuntu 4096 Jun 20 04:06 dialog

4 -rw-rw-r-- 1 ubuntu ubuntu 2829 Jun 20 04:06 __init__.py

52 -rw-rw-r-- 1 ubuntu ubuntu 49360 Jun 20 04:06 LICENSE

4 -rw-rw-r-- 1 ubuntu ubuntu 412 Jun 20 04:06 README.md

4 -rw-rw-r-- 1 ubuntu ubuntu 164 Jun 20 04:06 requirements.sh
```

```
4 -rw-rw-r-- 1 ubuntu ubuntu 79 Jun 20 04:06 requirements.txt

4 drwxrwxr-x 3 ubuntu ubuntu 4096 Jun 20 04:06 vocab
```

The dialog directory contains subdirectories for each language you want the robot to speak, and contains the responses to our commands. We will use the `en-us` directory to put our US English responses, since that is what my speech system is set to. You may use `it-it` for Italian, and so on. The directories use the **IETF Language Tag** (Internet Engineering Task Force), which you can look up at `https://en.wikipedia.org/wiki/IETF_language_tag`.
Other examples are `de-de` for German, and `en-au` for Australian English.

We will create a file called `pickup.toys.dialog` and put our responses in it, one phrase per line. We can put multiple dialogs and multiple skills into a single skill category, but we will just put the `pickup toys` command in this example.

We also have the `vocab` directory. This directory contains our intent phrases. These are identified by a `.intent` file. We need to create a `PickupToys.intent` file in the `vocab/en-us` directory and put our command phrases we wrote in it:

```
~/Mycroft-skills-1/skill_pickup_toys/vocab/en-us/PickupToys.intent

Lets clean up this room

Put Away the toys

Pick up the toys

Pick up all the toys

Clean up

Put those away

Put this away

Time to clean up

Who made this mess

mess

toys

clean
```

You will also need to clear out the old `.voc` files that you got by copying the template. We can insert just the key words rather than the entire sentence, and the Mycroft Intent Engine will still activate this skill.

Now we can populate the Python code that will activate the command to the robot. We need to edit the `__init__.py` file in the `skill_pickup_toys` directory that we copied from the template.

We are going to import our parts from Mycroft (IntentBuilder, Mycroft Skill, getLogger, and `intent_handler`). We also import rospy, the ROS Python interface, and the ROS standard message `String`, which we use to send commands to the robot by publishing on the `syscommand` topic:

```
from adapt.intent import IntentBuilder
```

```
from mycroft.skills.core import MycroftSkill from
mycroft.util.log import getLogger

from mycroft import intent_handler

import rospy # ROS = Robotic Operating System

from std_msgs.msg import String # ROS string format for messages

__author___= 'fxgovers'
```

This is the logger for Mycroft so that we can log our responses. Anything we put out to stdout, such as print statements, will end up in the log, or on the screen if you are in debug mode:

```
LOGGER = getLogger(__name__)
```

We set up the publisher for our `syscommand` topic in the ROS. This is how we send commands to the robot control program via the ROS publish/subscribe system. We will be publishing commands only, and the only message format we need is **String**:

```
 pub = rospy.Publisher('/syscommand', String, queue_size=1000)

# define our service for publishing commands to the robot control system # all
our robot commands go out on the topic syscommand

def pubMessage(str):

 pub.publish(str)
```

Our Mycroft skill is created as a child object of the `MycroftSkill` object. We rename our skill object class to `CleanRoomSkill`:

```
 class CleanRoomSkill(MycroftSkill):

 def __init__(self):

 super(CleanRoomSkill, self).__init__(name="PickupToys")
```

I can't explain why Mycroft has both an `init` method and an `initialize` method, but we need to follow the template. These commands set up the intent in the Intent Builder part of Mycroft and register our handler when any of our phrases are spoken. We refer to the dialogs we built earlier with `require("CleanRoomKeyword")`, so be careful that all the spelling is correct.

```
def initialize(self):

 clean_room_intent = IntentBuilder("PickupToys"). \
 require("PickupToys").build()

 self.register_intent(clean_room_intent, self.handle_clean_room_intent)
```

This section creates our handler for when the system has recognized one of our phrases, and we want to perform the action for this command. This is where we kick off the publish command to the robot's control program via the ROS using the `pubMessage` function we defined earlier:

```
def handle_clean_room_intent(self, message):
 self.speak_dialog("clean.up.room") pubMessage("PICK_UP_TOYS")
```

The rest of the program is just housekeeping. We need to define a stop handler and finally define a function to create the instance of our skill:

```
def stop(self):
```

```
    pass

def create_skill():
    return CleanRoomSkill()
```

In order for our skill to work, we need to copy our directory to `/opt/mycroft/skills`. From there, we can test it in debug mode. Remember to start the ROS Core service (**roscore**) first to receive the ROS messages.