15-494/694: Cognitive Robotics

Dave Touretzky

Lecture 11:

Speech Generation and Recognition

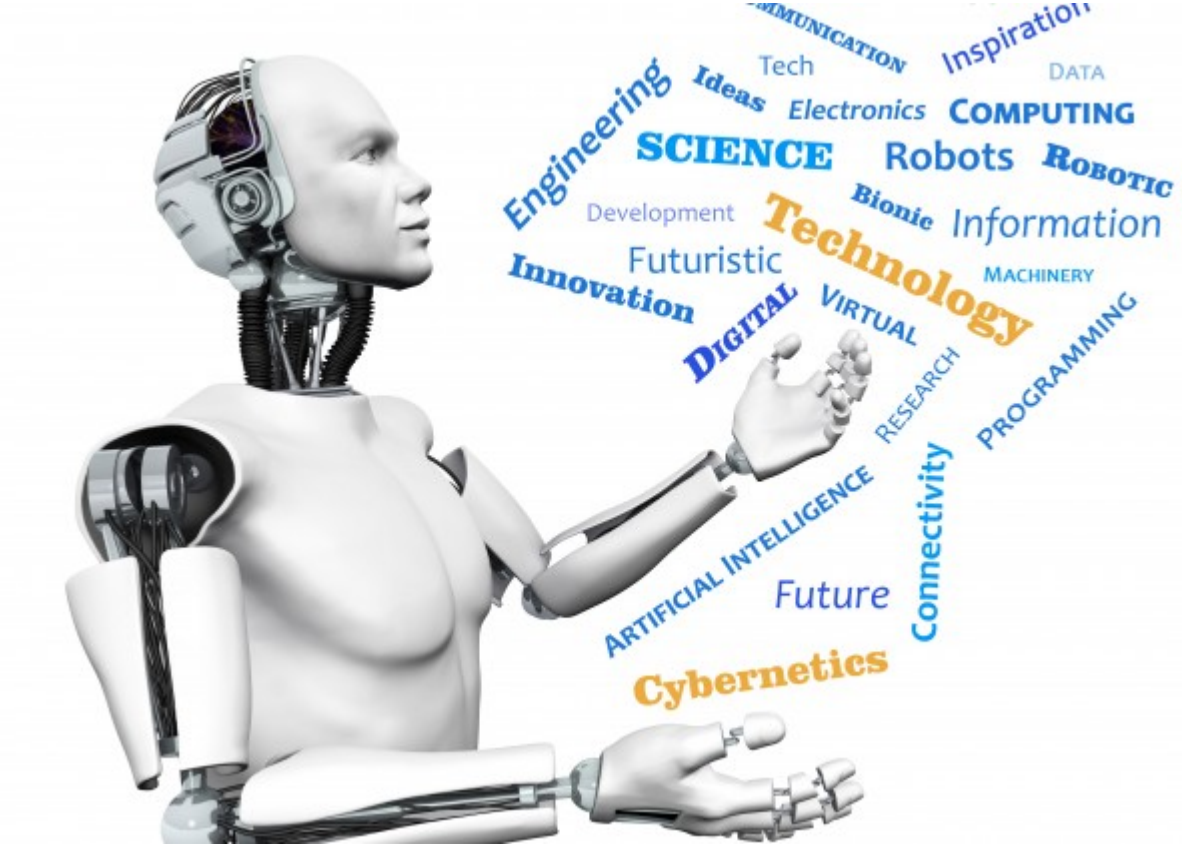


Image from http://www.futuristgerd.com/2015/09/10

Speech Generation

- We use the Google Cloud Text to Speech API to generate speech for VEX AIM.
- Parameters are defined in actuators.py

```
tts_voice = texttospeech.VoiceSelectionParams(
        language_code="en-US",
        name="en-US-Journey-F",
        ssml_gender=texttospeech.SsmlVoiceGender.FEMALE
    )
```

- Requires Google Cloud credentials. If not available, revert to gTTS package which uses Google Translate’s speech facility.

Google Cloud Text to Speech

- Info at https://cloud.google.com/text-to-speech
- Multiple voice models
  - Basic
  - Studio
  - WaveNet
  - Neural2
  - Journey (now “Chirp HD”)
  - etc.
- Some models allow control of speech rate and pitch.

SSML

- Speech Synthesis Markup Language
- Can be used to control pronunciation of things like acronyms or numbers
- Can be used to insert pauses where needed
- Not currently used in vex-aim-tools but might be in the future.

“Say” Node

- Constant case:

```
Say('hello there') =C=> next
```

- Variety (will choose at random):

```
Say([‘hello’, ‘hi’, ‘howdy’]) =C=> next
```

- Event-driven case:

```
Compute() =SayData=> Say() =C=> next
```

- Subclassing “Say”:

```
class SpeakBattery(Say)
```

SpeakBattery

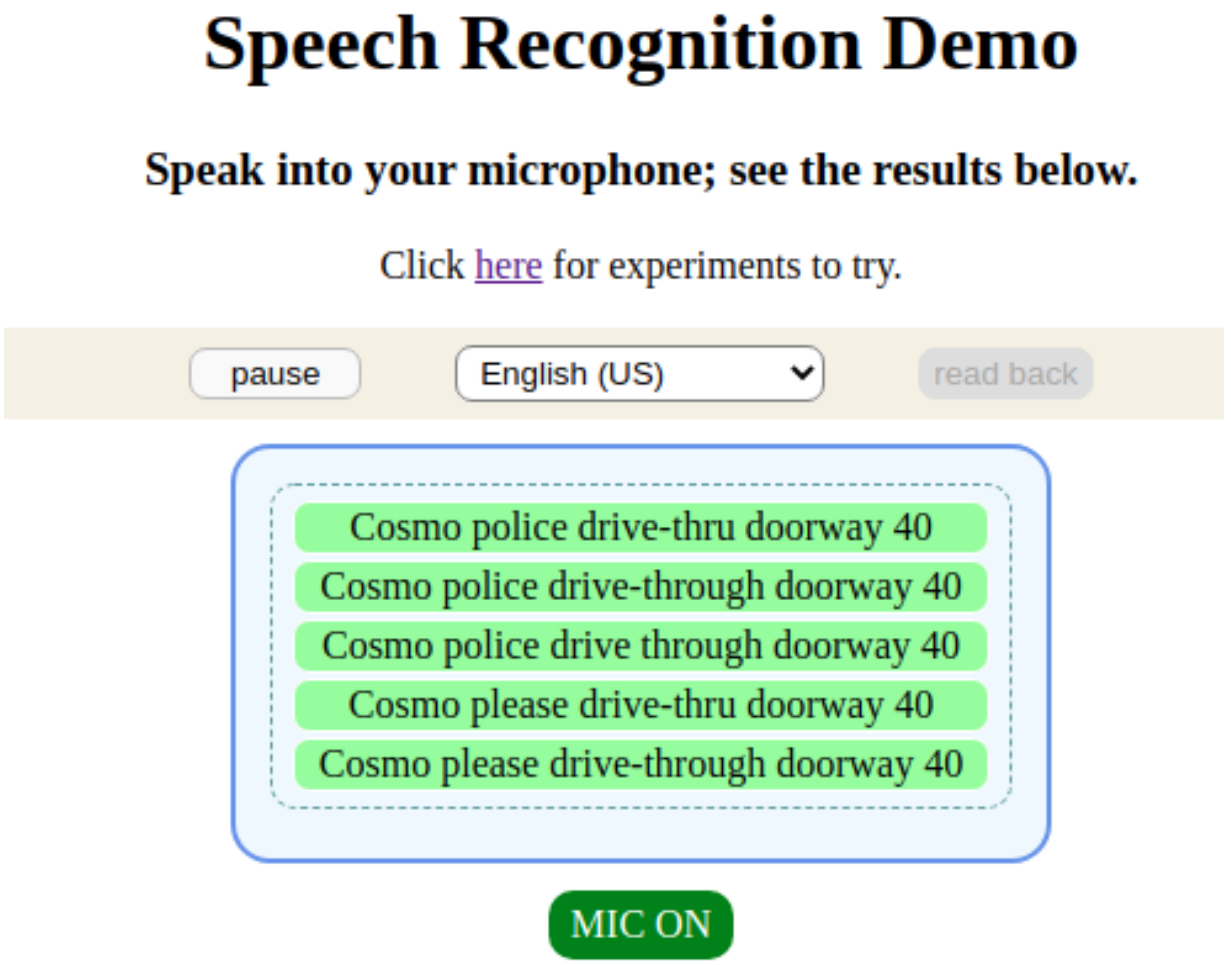
```
class SpeakBattery(Say):

  def start(self,event=None):
    cap = self.robot.battery_capacity

    self.text = f'battery capacity {cap} %'

    super().start(event)
```

Speech Recognition

- VEX AIM has no microphone
- Use the laptop's mic or a USB mic
- Recognition via the Google Speech API
  - Must have network access to function.
  - Biased towards conversational English, not arbitrary robot commands
  - Accuracy in 2025 is quite good.
- Sometimes uses special characters we don’t want, e.g., “15 degrees” is transcribed as “15°”.

Demo: Google Speech API

https://www.cs.cmu.edu/~dst/SpeechDemo



Hearing Our Own Speech

- To avoid the robot hearing its own speech, the Say node temporarily disables speech recognition before speaking.
- It re-enables recognition when the speech is complete.
- This process is imperfect. Mistakes will be made.

Declining Speech Recognition

Speech recognition is turned on by default.

To turn it off: use `speech=False` in StateMachineProgram.

```
class VEXCommand(StateMachineProgram):
  def __init__(self):
    super().__init__(speech=False)
```

When To Listen

- Microphone is always on
- We could use a wake word to indicate we're addressing the robot.
  - “Celeste, please grab a barrel”
- You've seen this trick before:
  - “Alexa, ...”
  - “Hey Siri, ...”
  - “OK Google, ...”

The =Hear()=> Transition

```
dispatch: Say('What now?')


dispatch =Hear('celeste turn left')=>
  Turn(90) =C=> dispatch


dispatch =Hear('celeste drive forward')=>
    Forward(50) =C=> dispatch
```

String Matching

- Convert everything to lowercase
- Remove all punctuation
- Normalize homophones

Homophones

- “Thesaurus” data structure defined in aim_fsm/speech.py
- Words:
  - cozmo ← cosmo, cosmos, cosimo, …
  - right ← write, wright
  - cube1 ← q1, coupon, cuban
- Phrases:
  - cube1 ← cube 1
  - paperclip ← paper clip

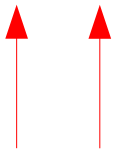
Regular Expression Matching

- Uses the Python re package
- Example: optional words

  'celeste ?(please|) drive forward'

- Be careful about spaces!
  - Example: scanning for keywords:

    'celeste .* grab.*'



spaces on both sides of .* will be a problem
if the .* matches the null string

Checking the Match Results

- When a =Hear=> transition fires, it offers a SpeechEvent to the target node(s).

- The SpeechEvent contains three items:
  - **string:** the string that was matched
  - **words:** list of words in the string
  - **result:** the match result from re.match
    - contains the groups defined by ( )

Extracting Groups (1)

```
from aim_fsm import *


class Speech1(StateMachineProgram):

  class Heard(Say):
    def start(self,event):
      obj = event.result.groups()[1]
      self.text = 'I will grab %s' % obj
      super().start(event)
```

Extracting Groups (2)

```
$setup{
  loop: Say('what now')
  loop =Hear('celeste ?(please|) grab a
    (barrel|ball)')=>
       self.Heard() =C=> loop
  loop =Hear=> Say('Pardon me?')
          =C=> loop
}
```

Dialoging with GPT-4o

- Instead of parsing user utterances with regular expressions in HEAR transitions, we can let GPT-4o do that work.
  - Much better strategy!
- Now the problem is how to get GPT-4o's understanding back into our program logic.
  - Use #hashtag tokens for actions
  - How else might they be used?

Dialog

- Dialog requires access to a knowledge base and a mechanism for retrieval.

- What's in the knowledge base?
  - The world map
  - The robot's recent actions and plans
  - Recent object references
    - Necessary to resolve "it", e.g.:
      - "Do you see an orange barrel?"
      - "Pick it up."

What Else Is Needed?

- Celeste has no sense of time. We can't refer to "the object you saw 5 minutes ago".
  - Add timestamps to the prompts?
- Our current Query-Response structure is too inflexible.
  - The robot must wait for our next input.
  - What if we want the robot to initiate action on its own?