Kyle Lewis / Alina Vorobyova
124 Final Write-up

Motivation and Overall Concept

Machine learning is a form of mathematical optimization which has grown rapidly in industry within the past few decades. The concept sees ubiquitous use in applications of image recognition, recommendation algorithms and speech recognition, among others. Our goal with this project was to learn the current limitations of existing speech recognition software, using such software to interface with the arduino and ultimately to control a car with our voices. In doing so we are experimenting with how easy it is to apply existing speech recognition software in small applications, as the technology becomes more and more accessible these sorts of applications will become possible for wider distribution, “google home” for example.

Our baseline goals were audio input to an Arduino unit, and speech recognition through a freely provided server platform. With that achieved,we moved on to outputting a response based on our voice commands; for this we plan on building a car which will be run by a predefined set of voice commands.

Functional Definition (Input, Processing, and Output)

The input audio signal was at first recorded by a laptop microphone, and later we used a small directional microphone with a USB interface to act as the laptop’s primary recording device. This is so we could isolate commands from outside noise. After audio is captured, the voice recognition software we are using, “BitVoicer”, provides a user interface which allows us to monitor recognized words and the activity of the server, more details on BitVoicer software later. No amplification or modification of these audio signals is necessary. This is all an alternative to using an electret microphone input on the arduino directly, which ended up not working.

Having our command signal in digital form, it is sent to the BitVoicer server for translation. BitVoicer is a free software provider with their own server based neural net, which has libraries for Arduino and can process digital signals to be recognized, freeing the Arduino from the computationally heavy task of voice recognition. The BitVoicer server works with a set of user built “sentences” and an activation word; upon receiving the activation word and recognizing speech from one of the sentence trees we define, a string command is then sent to the Arduino over the Serial interface to perform output actions.

Sensors

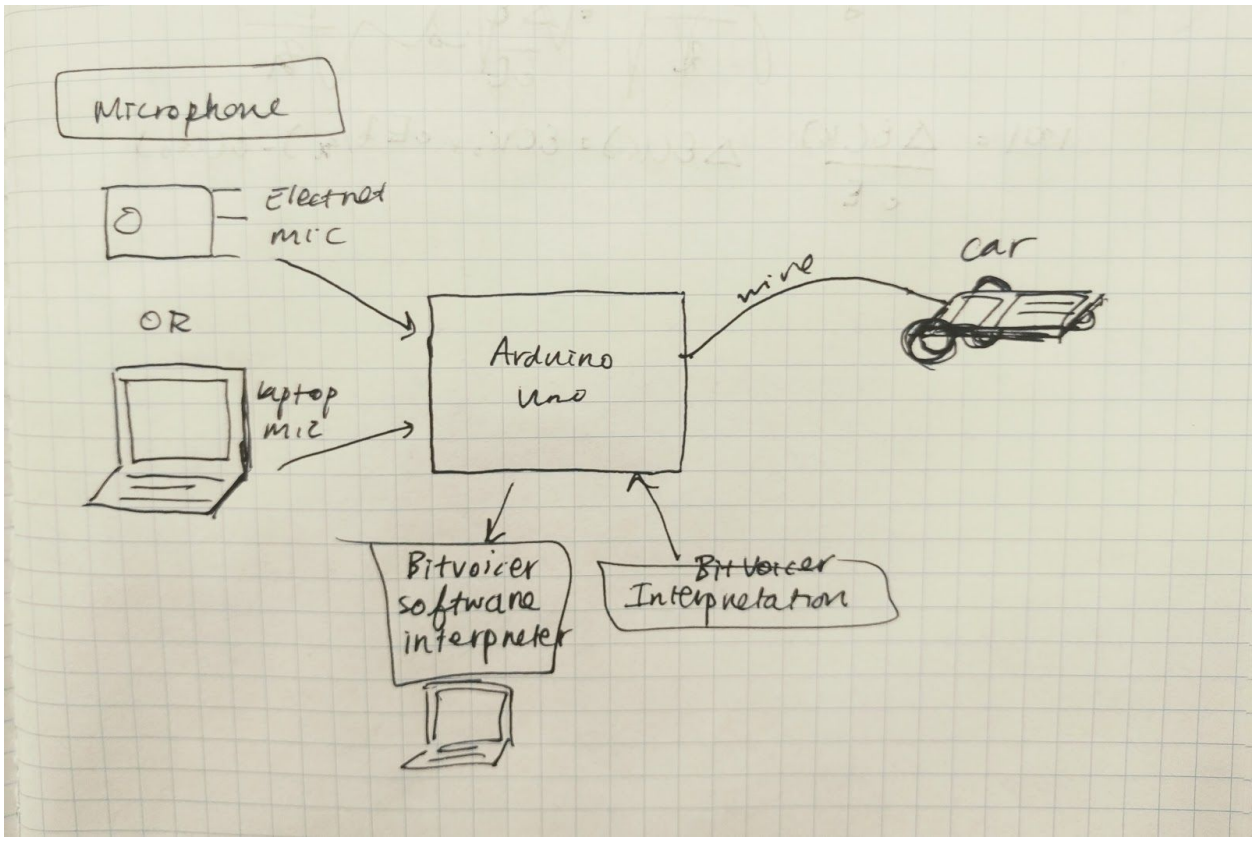
- The microphone is an “Eberry Plug and Play”
- The above is amplified internally and no circuitry is required to do so.
- Our Car includes a proximity sensor to prevent it from running into walls and such, but the code to do so is not yet written. We’re using analog output IR sensors from the lab.

Mechanical Considerations

The mechanical requirements of the project are found mostly in the car, we ordered a kit to simplify and save us some time. The car has an on-board breadboard and is wired - using ribbon wire which is lighter - to the arduino and motor shield. The wheels of the car have decent grip on the linoleum of the lab floors, but there have been occasions where they slip after accumulating enough dust and such. For finer turning controls these tires should be replaced with something more grippy. We've seen groups use inversed tape on their wheels, perhaps that would have worked. The motors on our kit are simple DC motors which spin quite well but don't give the most torque, "hanging" the wires up and away from the car has become necessary to avoid most of the tug which the wires provide, as this alters the motion of the car significantly. At the very end, we have some photos of the final result.

Electrical Considerations

Powering the microphone or laptop, and wiring it for USB communication was simple enough. The wires to the breadboard are cut quite long and taped down to the back end to prevent most cases of tangling with the car or otherwise. To control the motors we used the AF Motor Shield, which can be used to drive two outputs to the breadboard on the car. Please refer to final photos at the end of the document.



Interface

We used the laptop and attached microphone as our capture device, for which only power is required. The car takes two sets of pins for each motor, as well as a few LED's and three extra for an IR distance sensor, which we did not get to implementing in the code. We had hoped, or perhaps planned for an expansion, to implement RF communication in which case would require a data output pin as well. Communications with BitVoicer are done with the serial input, so we upload the control code first and then hand over the serial-USB connection to BitVoicer to monitor it.

Software

Achieving our base goals we were able to program entirely in C, in the Arduino IDE and used the library for BitVoicer which is provided. Key subroutines include:

- Utilizing the BitVoicer library API to use their serial protocol and transfer commands to and from the Arduino, this required a minimal understanding of their data frame and flag structure, essentially we test commands for containing a string, and then compare them to perform actions.
- The set of output commands to send to the car upon receiving data from the Server, setSpeed writes to the car's motors using the AF Motor Shield for a given duration and orientation, depending on the command ("turn" will only apply to one given motor, for example).

These processes are all procedural and case based, if you wanted to stop the car with the IR sensor an interrupt would likely be necessary.

Testing

Testing was done at all of the stages outlined above, input processing and output. Testing the electret microphone lead to our changing to the laptop microphone, and then later the small directional microphone. The issue here was audio levels jumping between zero and 100 on BitVoicer's software.

We tested BitVoicer's software with LED's, using some of the operating function checks which are provided in their API. The next step was to actually test commands relevant to the car, for which we wrote a separate set of code so we could just test car motions with the keyboard. We've tested in noisy environments, and found that the project works quite well with the directional microphone in these environments.

Safety

No parts require high current or voltage to drive them, and the bulk of the project exists in code. We did not accidentally create an evil AI(Ultron)! The car is powered directly by the Arduino so we didn't need to worry about batteries.

Parts Required and Reusability

We did not use an electret microphone, but instead used the laptop microphone, and later upgraded to a basic microphone that can be attached via USB. For processing and sampling capabilities we just required the Arduino Uno. Here is the car kit that we used for the project: https://www.amazon.com/gp/product/B01BXPETQG/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1

We were hoping to use RF sensors, but ended up opting out of using those. Aside from these main materials, we used some LED lights and lots of ribbon cable.

Results and Issues

Our final result did not use RF sensors, so our ideas for expansion were a bit inaccurate. We tried getting the RF sensors working for a couple of sessions, but after doing some research and hearing about other groups' attempts at using RF sensing from TAs, we decided not to follow that route(apparently it can take months to get working). Instead, we opted for using very long ribbon cable to connect the car to the arduino and laptop. We also attached LED lights that were useful for debugging purposes since the motors would sometimes not work, but this way we would be able to tell whether or not the command was being sent.

This relates to one issue that we ran into in which the motors would simply not work to their full capacity. We identified a couple of reasons, namely that the screws holding the wheels were actually a bit long, to the point to which they would be touching the wheels so as to cause extra friction. We even noticed some shavings where the screw ends will shave pieces of the plastic from the wheel. We were able to resolve the issue by un-tightening the screws, which although made the car less stable, ultimately allowed for better motion. Furthermore, the floor would not provide enough resistance, so at times, the wheels would slide instead of actually moving the car. One final problem was the wiring, because we had to use long wires, they would cause resistance to the car's movements. We more or less resolved this issue by choosing a light ribbon cable instead of the thick individual wires that we braided together. A better fix would have been to use bluetooth, but we did not have enough time to configure that in this case.

Perhaps the most time-consuming problem, after setting up the BitVoicer-Arduino interface of course, was "calibrating" the DC motors. This entailed finding the values for time and voltage that corresponded to different angles and distances. This just took time, because as we changed the factors affecting the motors' performance(listed in the previous paragraph), that would change voltage and time values.

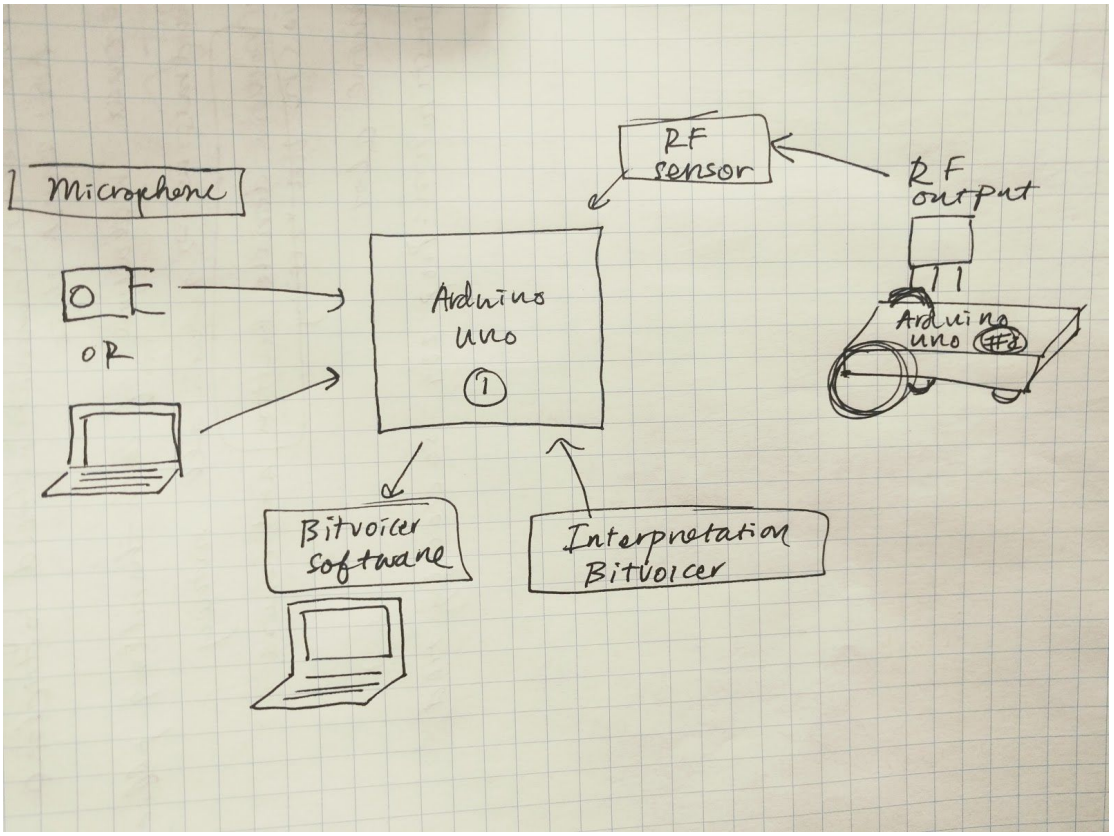
Finally, we attached a distance sensor to the front of the car. The idea was to get the car to avoid hitting obstacles in front of it, but unfortunately we only got so far as sending a warning red light when some obstacle is in front of the car.

Expansion

If we expanded upon this project, we would definitely implement RF communication, or Bluetooth, which we understand may work better. We could use one of the RF modules which we've found in the lab, and a second arduino uno would be mounted on the car to receive and parse the RF commands, likely using the library "VirtualWire RF", or a Bluetooth shield and the matching library. Cutting down on the wire interference would make the car work much better.

Changing out the motors would be another expansion option. Currently they are fast turning (low torque and low control) DC motors, we could change these out for motors which could be driven with a higher torque and slower speed, and at that point we could control the turns of the car to a greater degree.

Finally, fully implementing the distance sensor would be interesting. It might be nice to add more than one so that the car could also prevent all kinds of collisions.



De-Scoping

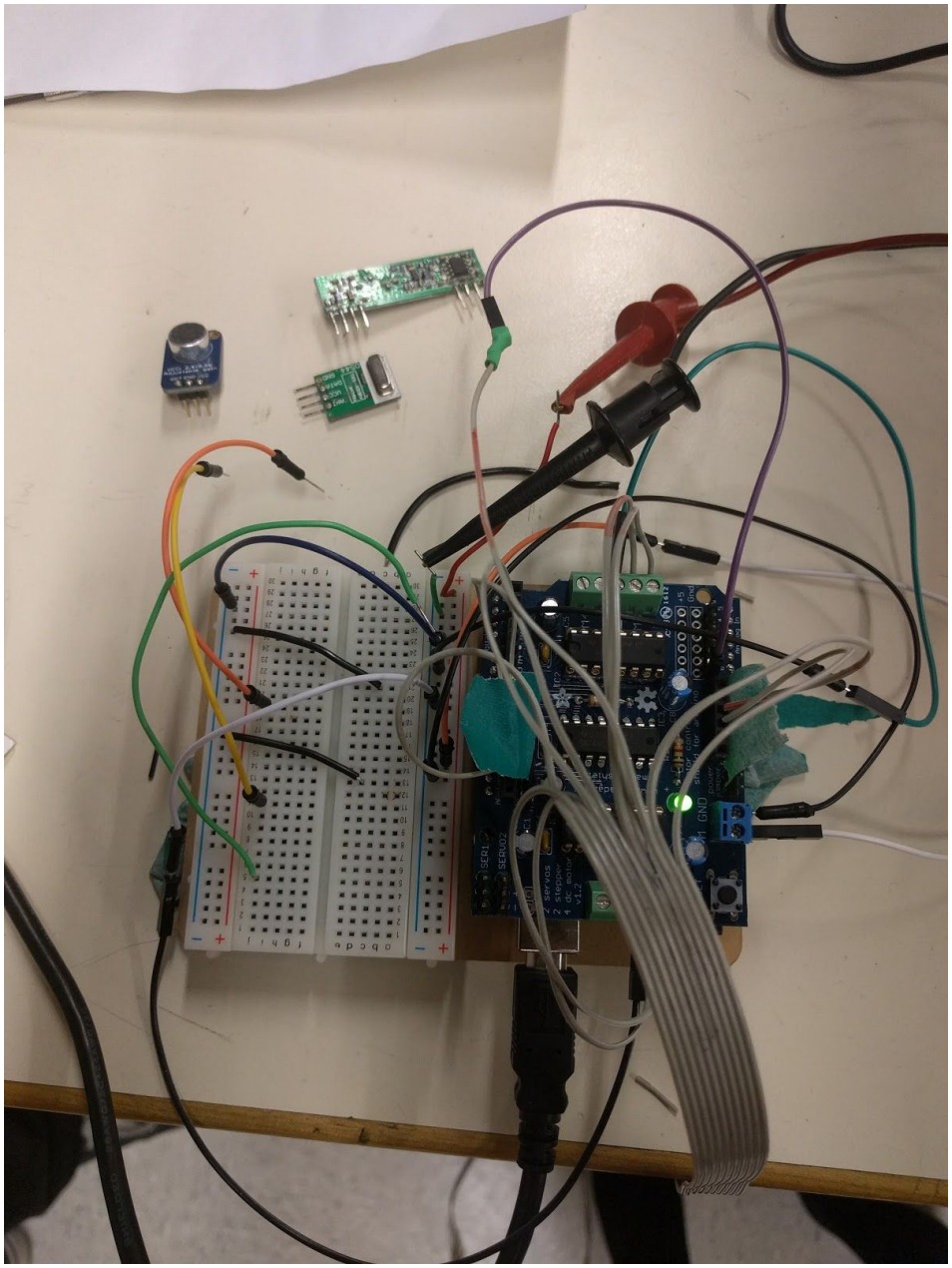
At many points in the project there are libraries available to simplify things, and these are our go to for De-Scoping. For audio input we ended up using a laptop USB input microphone, which simplified our audio issues quite a bit. We can, and are presently using the laptop microphone for direct input to the server as the recognition step. For recognition we already use a library, but limiting our set of commands to 10 or

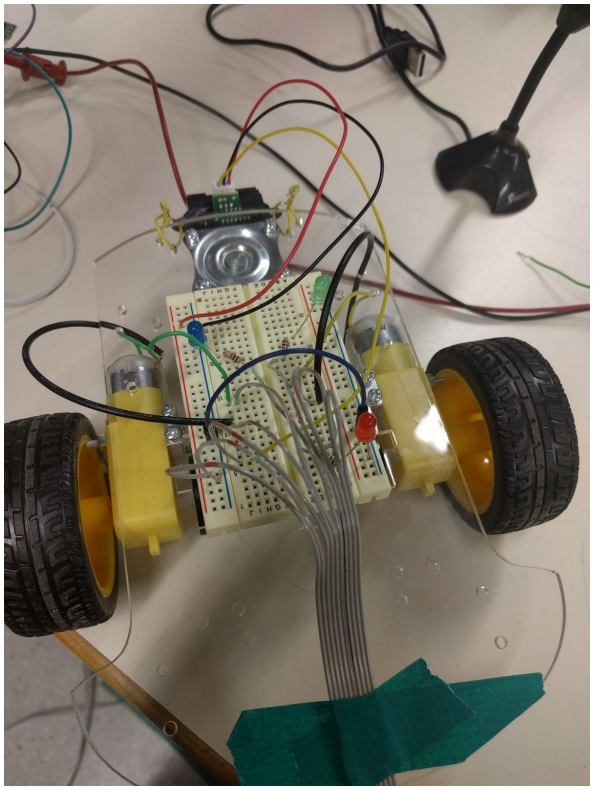
so, with a few variables like “left” and “right” allows us to test all of them given the time that we were allotted.

We didn’t need to use something other than the library, but our car was made as simplistic as possible in order to complete it, and to ensure we could control it well. Adding additional outputs on the car also had the problem of adding more wires to the arduino, which dragged the car more, so we avoided this.

Turning In

We demonstrated our voice controlled car during the presentation on friday, and have provided some notes in above sections involving what went wrong and where.





(older version) of the BitVoicer GUI. Here we could define sentences to be searched for by the server, and link them to string commands to be sent to the Arduino