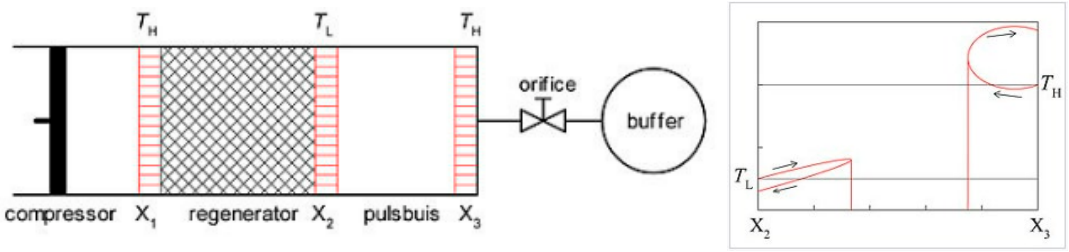
Daniel Lacey

Atlanta, GA |dan@laceys.com | http://dan.laceys.com/ | (518) 545-9276

3d Printed Pulse Tube Refrigerator (Still in progress)

A pulse tube refrigerator is a Stirling cycle refrigerator system. The goal of the system is to create rises in fluid pressure while limiting fluid velocity allowing heat to be ejected from the system. This is done by using flow resistance positioned after the heat ejection section. To allow the system to reach lower temperatures on the cold side a temperature gradient must exist between the compressor and heat ejection sections, this is done with a regenerator. The goal of a regenerator is to keep the compressor section hot and the heat ejection section cold. Good regenerator properties are being able to absorb heat well through convection, poor heat conduction along the direction of fluid velocity, and the regenerator shouldn't resist the flow of the system.



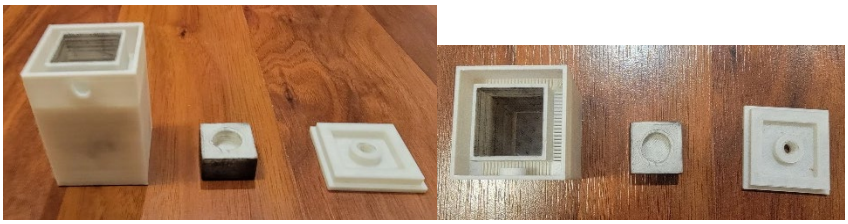
My goals for this project were to design the pulse tube refrigerator: to limit the number of non-3d printed parts, make the assembly as easy as possible and avoid any post-printing processes (like heat treating, drilling, epoxy coating, etc…), limit the overall cost of the system, and achieve a 30 degree Celsius temperature difference.

Note: Pulse Tube Refrigerators can have much larger temperature differences than 30 degrees Celsius, 30 degrees is just an initial starting goal.

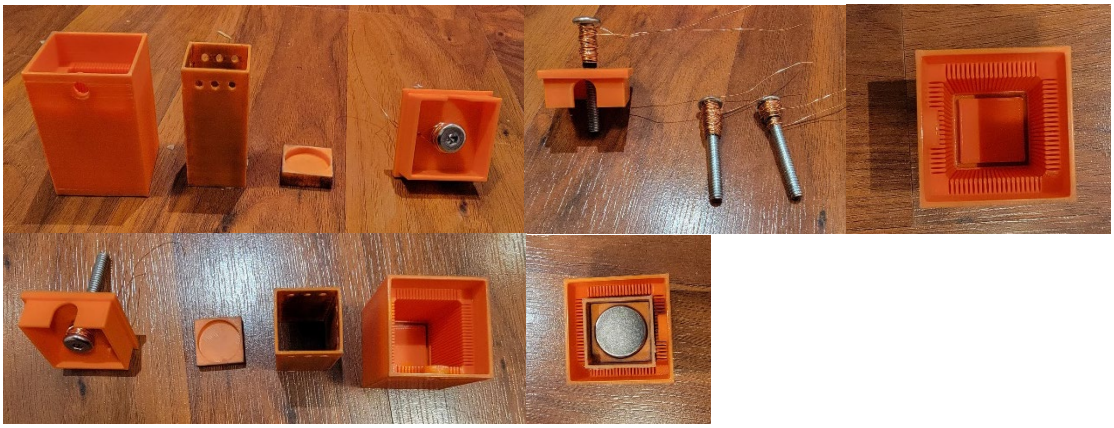
The biggest design challenge of the project was to create a sealed high-compression piston system. The initial test to was test the tolerances of the 3d printer in creating circler piston systems. It was found the 3d printer (Ender 5 Pro) had a tolerance of ±0.005 inches for the print settings I was using. Limiting the layer height of the print greatly improved the smoothness of the prints reducing friction. Multiple lubricants were tested to help reduce friction; the best was graphite dust.

The first and second prototypes had square pistons. This was done to reduce the tolerance between the piston and the piston tube. Any curve being printed was approximated by using hundreds of small straight lines. This caused greater tolerance between the circle piston and piston tube, therefore a square piston design was used instead.

Prototype 1



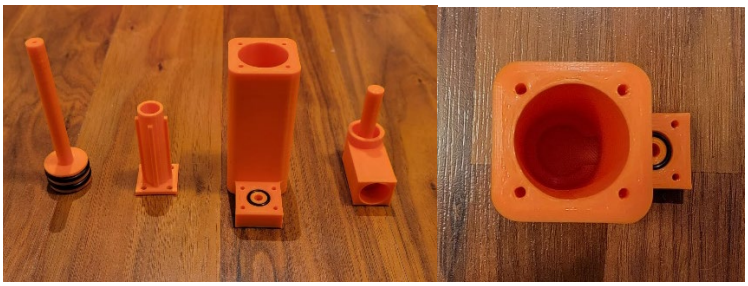
Prototype 2

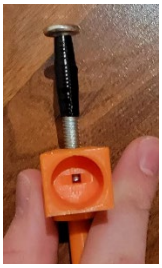


The two prototypes use electromagnets to drive the piston to simplify the mechanical component of the design and to eliminate the need for an o ring or any type of sliding seal. The electro-magnet was controlled with an L298n (H bridge circuit) and an Arduino. The drive system worked and could generate a notable pressure spike but required a very large amount of current to work. Even though the piston and piston tube had a small tolerance there still would be losses due to improper piston seal.

The third prototype was designed to achieve higher compression ratios with the piston. The piston was switched to a circle design and uses o rings to seal the piston. A small passageway at the bottom of the piston tube connects the piston tube to the regenerator. The regenerator is attached to the piston tube with No.6 bolts and uses an o ring as a face press seal to make the connection between the regenerator and piston tube airtight. The regenerator then connects to some copper pipe which acts as the heat ejection section, which then connects to the needle valve. The needle valve has a cylindrical section that inserts into the heat ejection section to decrease the distance between the fluid and the copper pipe, increasing heat flux out of the system. The flow restriction of the needle valve is controlled by screwing in a bolt down to block off a passage for the fluid.

Prototype 3





Future tests to run on the latest prototype: find max temperature difference with no thermal load, test multiple regenerator lengths and materials, and test multiple levels of flow restriction.

My future design goals for this project are to design a drive system to drive the piston, test multi-material regenerator designs, and combine the regenerator tube and piston tube into one print after finding the optimal regenerator design.

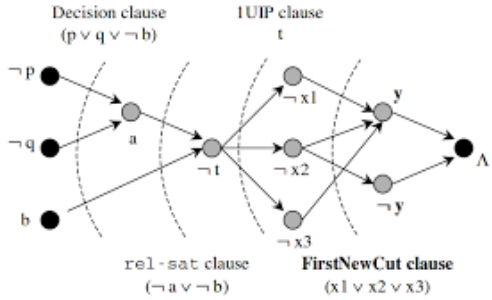
This project is still in progress (photos are not the most recent), updates to this project are coming soon.

SAT # Solver (2020 - 2022)

SAT solvers find solutions for Boolean Satisfiability Problems (aka Boolean Expressions) which can express many problems in computer science, engineering, logistics, scheduling, number theory, and many other fields. The Boolean Expressions are transformed into CNF (Conjunctive Normal Form), where smaller Boolean expressions, known as clauses, are all connected with operators. The clauses are Boolean expressions where variables are all connected with "OR" operators. SAT # solvers are a type of SAT solvers that finds all solutions to the given Boolean Satisfiability Problems. For example:

$$x_1 \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_4 \vee \overline{x_6} \vee \overline{x_7}) \wedge (x_1 \vee x_2 \vee \overline{x_3} \vee x_5 \vee x_7).$$

Currently, the industry standard SAT solvers are based on the CDCL (Conflict Driven Clause Learning) algorithm. Which guesses a variable's state (0/False or 1/True). Then CD propagation occurs, and the variables' states are compared to the clauses to find what other variables' states are implied to be true for the full expression to be true. If CD propagation leads to a variable being implied to be both true and false, then CL occurs. CL finds the variables' states that implied the conflict (aka already implied variable) to generate a new clause to avoid that conflict again and backtracks the earliest variable in the new clause.



An alternative to CDCL is the Flash algorithm. The Flash algorithm is like the CDCL algorithm but instead of guessing single variables blindly one at a time, the Flash algorithm solves a cutset of the SAT problem to choose what variables' states to set. Conflict Driven propagation and Clause Learning are also used to expedite the solving. A cutset is a section of the overall SAT problem with a sub-set of variables and clauses from the original SAT problem. Only clauses where every variable in the clause is in the sub-set apply to that cutset.

These cutsets are solved for all possible solutions. These solutions are then modified/reduced to only the variables' states that provide useful information for speeding up the search. Multiple variable groupings are generated and solved to find which has the smallest branching factor. Variable groupings are generated by using a dynamic selection function based on the initial variable selection seed, which are the variables found in each clause. The dynamic selection function iteratively selects clauses with the minimum number of new variables to the current variable grouping.

After variable grouping is generated, the variables are mapped from the variable grouping set to values between 1 and the size of the variable grouping. This format is needed for the cutset SAT # solvers. The mapping of the variables is determined by the frequency of a variable in the cutset's clauses. The greater the frequency of a variable lower the number it would be mapped to. This results in a significant speed up in the cutset SAT # solvers.

The default cutset SAT # solver used in this program is DTD (Dead Tree Decomposition). The DTD algorithm focuses on calculating the variable state combinations that make a given clause false. It does this for every clause, and if any combinations overlap it skips to the next unique variable state combinations for that given clause to avoid repeating work. DTD is the default solver because it has a very quick and consistent cutset SAT # solver regardless of how many solutions there are to the cutset. There are other cutsets SAT # solvers that can be faster than DTD only when there is a smaller number of solutions. As mentioned above the solutions are modified/reduced to only the variables' states that provide useful information for speeding up the search. This is done by only returning the bits that were in both the low and high values of the bound values. The CD propagation is applied to all reduced solutions generated from the cutset, if there is no conflict then the solution becomes a branch, and the process repeats until all branches die or all solutions are identified.

During this project, I did extensive testing on which data structure would work best for the DTD solver. The main concern was speed. The data structure had to hold a large numerical value while being able to access binary bit values quickly. Some of the data structures investigated were arrays of Boolean variables, an array of integers confined to values 0 or 1, Bitset (C++ only), and large integers. It was found that arrays of integers confined to values 0 or 1 resulted in the fastest solutions (C++ testing and more details about the results can be found on my GitHub).

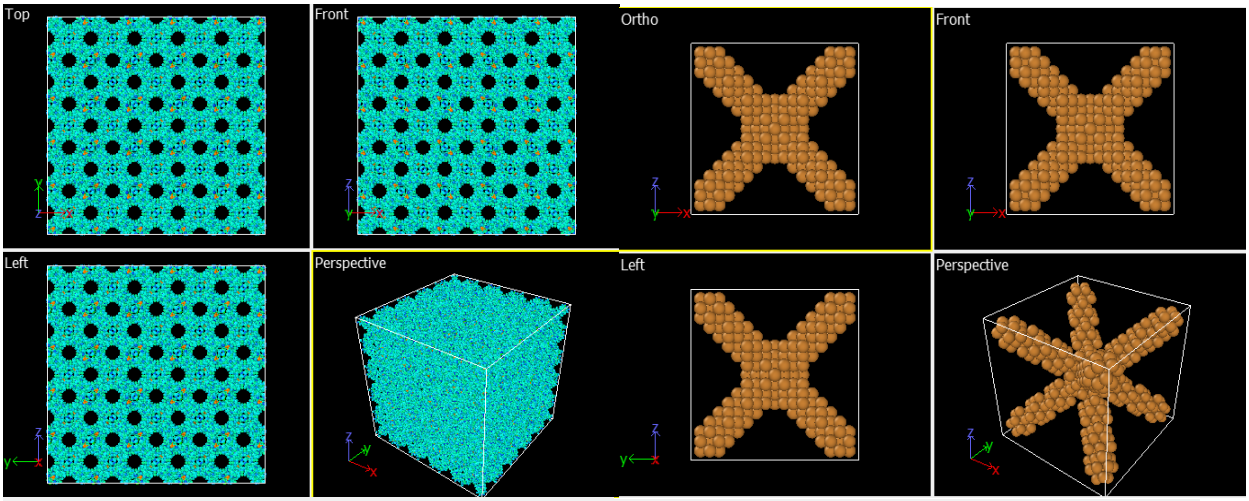
I also ran many tests using different languages and compilers. For testing, I only focus on the DTD solver. Once again, my main concern was speed. I tested python (basic), python (pypy), C++ (g++), and C++ (C-Lang). I found that Python using pypy was the quickest. Pypy is a JIT (just in time) compiler for Python, allowing code to have a comparable speed to non - interpreted languages. The original expectation was C++, an AOT (ahead of time) complied langued, would beat a JIT compiler. C++ ran notifiable slower even with all optimization flags on the compiler turned on. (Testing and more details about the results can be found on my GitHub). I believe the reason that the JIT compiler was better suited to this task is due to the scale and complexity of the program. JIT compilers, unlike AOT, compilers can perform pogo (profile-guided optimization) which only becomes advantageous for large programs. JIT compilers also perform better at pseudo-constant propagation, indirect-virtual function inlining, etc.

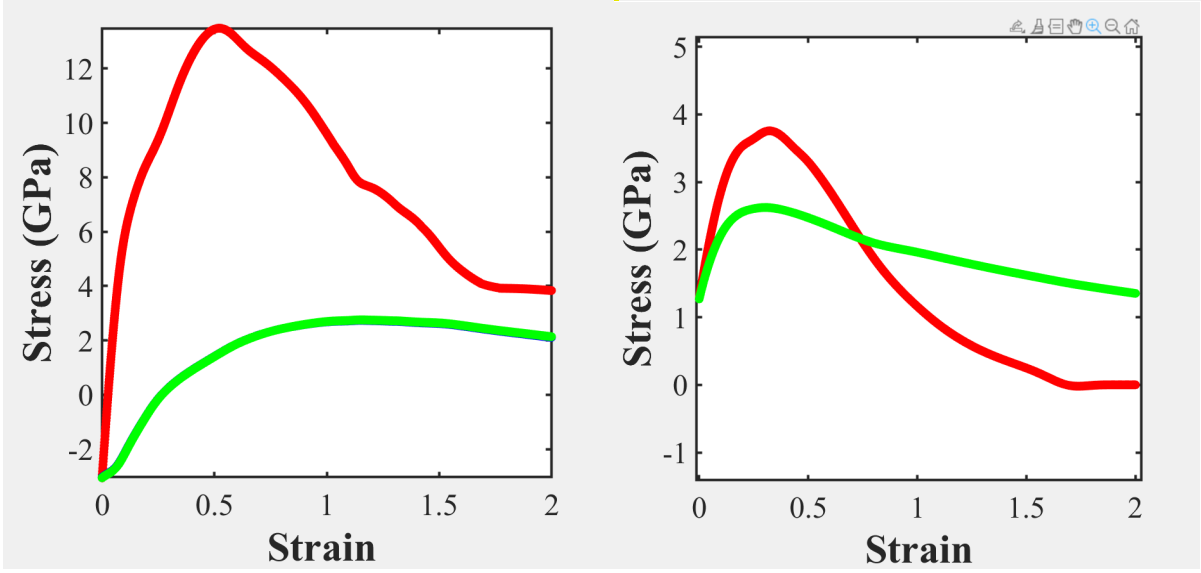
This is still an ongoing project and additional improvements still can be made. The main bottleneck of the code is the cutset variable grouping. Four possible solutions are GCN (Graphical Convolutional Networks) and GNN (Graphical Neural Networks), pre-planned grouping, different greedy search algorithms, and coding the search algorithms to be GPU compliant. The next big improvement that could be made is to include clause learning in the algorithm. Allowing the program to learn new clauses from CD propagation conflict and possibly from dead cutset (cutset found to have no solutions). CD propagation can be improved upon by only propagating over the clauses that had a variable recently assigned.

Making the code able to run in parallel would increase the speed of cutset variable group searching significantly. This could be done in multiple ways. Either using multi-threading, multi-processing or making the code able to run on a GPU. Parameter tuning may also improve the speed of the program by selecting the optimal cutset's size, variable grouping algorithm, and SAT # solver for each branch. Finally, a computation time estimator would be a useful addition to the code.
The code can be found on my GitHub: FernandoRando1

Undergrade Research (2020 - 2022)

I have conducted undergraduate research under Ioannis Mastorakos on the modeling, manufacturing, and testing of nanoscale metallic composites and nanofoams. We have investigated numerous types of nanoscale metallic composites/nanofoams and different methods of manufacturing them. We have focused on chemically restructuring MOFs (Metal-Organic Framework). I have used LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator, a molecular dynamics program) to simulate the formation of the nanofoam and calculate the stress-strain curves of the nanofoam under compression and tension. Experimental nano-indentation has confirmed the simulated results. We also simulated geometrically similar structures and ran tension and compression tests to test how much the geometric structure affects the mechanical properties. I have been using Ovito (an open source visualization and analysis software for particle simulation data) and MATLAB as post-processing programs; Ovito to visualize the data and export data into usable filetypes, and MATLAB to graph the data.





Note: The Stress does not equal zero at zero strain due to internal stress. This does not affects any of the mechanical properties.

Artificial Learning (Reinforcement Learning)

Reinforcement learning is a type of machine learning based on rewarding successful behaviors and punishing unsuccessful behaviors. I developed four RL models (policy gradient, dqn (deep q network), double dqn, a2c (actor-critic 2)) and tested them with the cart pole environment model.
The pole cart environment model is designed to be a simple AI training environment where the AI controls a cart constrained to one dimension (left and right), with an inverted pendulum on top of it. The AI is scored based on how long it was able to keep the inverted pendulum upright and the goal of the AI is to maximize this score.
Due to the instability of the dqn and double dqn models during training (aka tendency for scores to drop significantly), I added a checkpoint feature. If the scores did not increase, it would reset the model to the last checkpoint and temporally increase epsilon (the probability of the AI choosing a random action) to prevent the AI from repeating the same mistakes that caused the scores to drop. This greatly improved the stability of the models.
Note: double dqn is much more stable than dqn, but still suffers from instability during training.
In the graphs below the blue bars are the run times of each run and the green line is the average of the previous 20 runs.

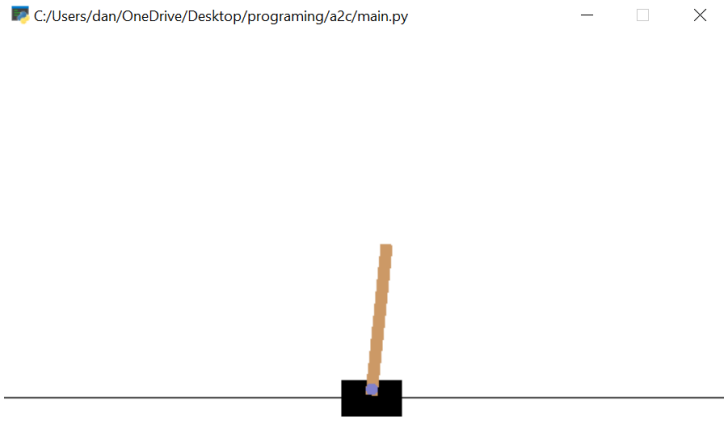
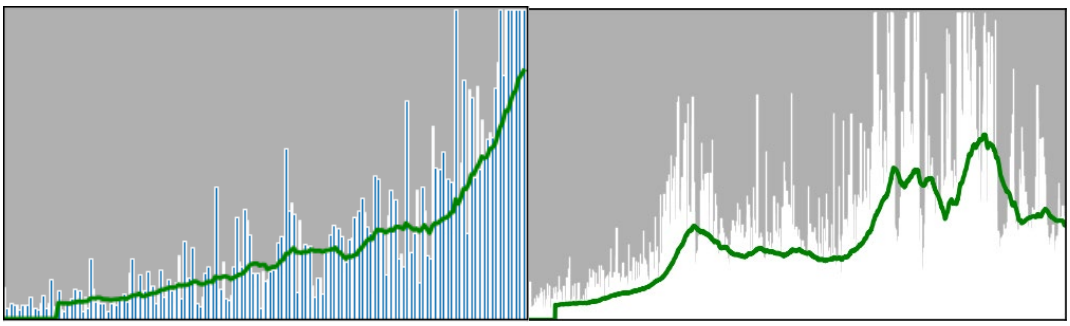


Image of cart pole environment model

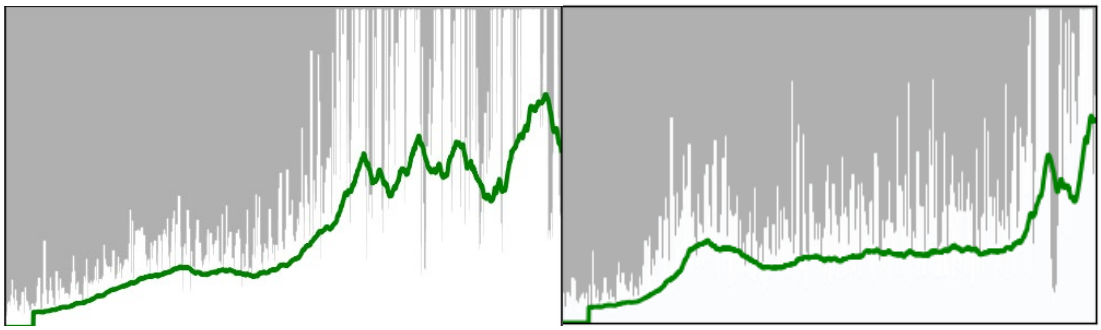
<u>Policy Gradient:</u>

<u>Deep Q Network:</u>



Double Deep Q Network:

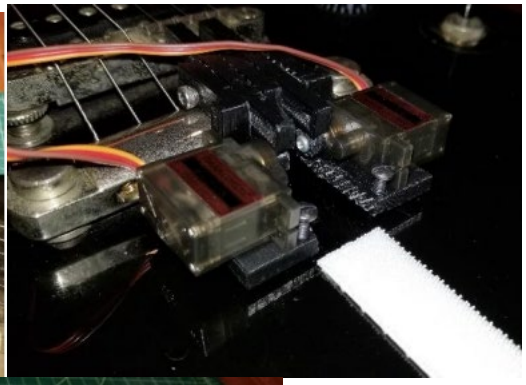
Actor Critic 2:

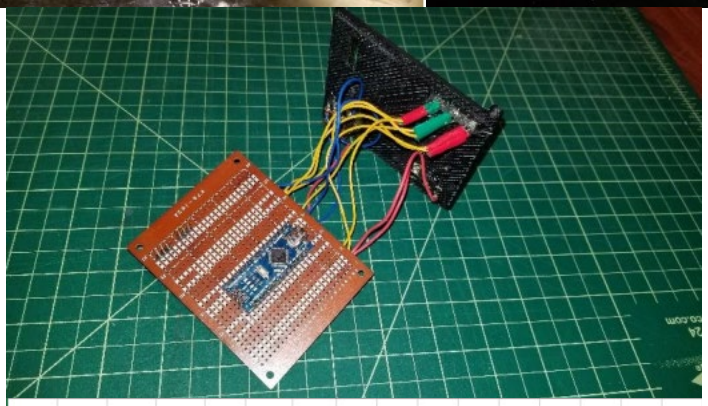


The code can be found on my github: FernandoRando1

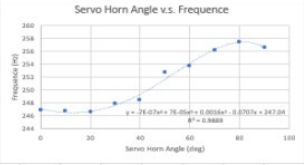
Arduino Controlled Guitar B/G-Bender

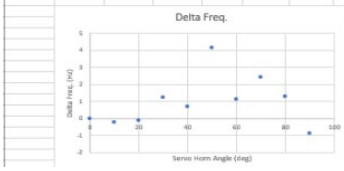
Arduino Controlled Guitar B/G-Bender (Spring 2020) As an independent project I built a b/g-bender for my electric guitar. A b/g-bender is a mechanism that increases the tension in the b and/or g string of the guitar, changing the pitch of the string. I designed and 3D printed a jig that securely grips the bridge of the guitar which has two servo motors mounted to it. The servos turn a small cam, raising one side of a lever arm attached to the jig, causing the other side of the lever to press down on the b or g string; therefore, increasing the tension of that string. The servos are controlled by a nano Arduino (microcontroller). The device has three modes: b string, g string, g, and b string. The modes can be cycled through by using the push button under the LEDs. The three LEDs indicate the following: power on, b string, and g string. The servos can oscillate their position to give a vibrato effect. The speed at which the servos oscillate their position is controlled by the potentiometer located in the opposite corner from the LEDs.

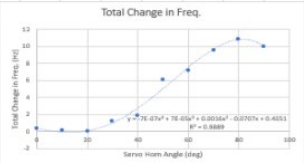




Servo Horn Angle (deg)	b-string Freq change test 1 (Hz)	b-string Freq. change test 2 (Hz)	b-string Freq. change test 3 (Hz)	AVG	Total Change in Freq.	Delta Freq.
0	247.6	246.4	246.9	246.967	0.333333	0
10	246.9	246.4	246.9	246.733	0.1	-0.23333
20	247.1	246.3	246.5	246.633	0	-0.1
30	248.2	247.8	247.5	247.833	1.2	1.2
40	248.8	248.8	248	248.533	1.9	0.7
50	253.1	252.2	252.8	252.7	6.066667	4.16667
60	253.9	253.8	253.7	253.8	7.166667	1.1
70	256.3	256.1	256.2	256.2	9.566667	2.4
80	257.5	259	257	257.5	10.86667	1.3
90	256.8	256.8	256.3	256.633	10	-0.86667
Min Freq.	246.6333					
Max Freq.	257.5					
Max Freq. (	10.86667					

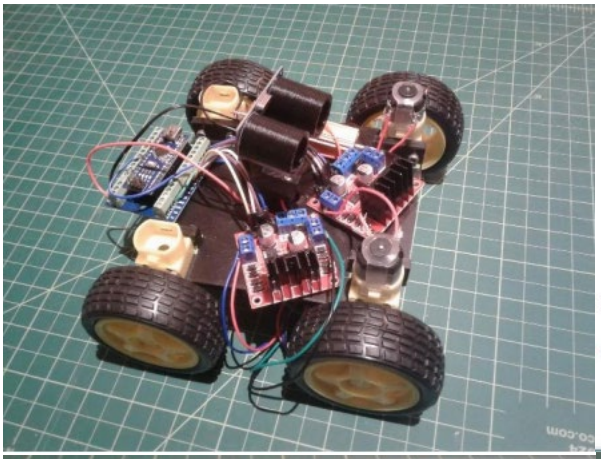






Mobile Arduino Based Scanning Robot

As an independent project, I decided to learn more about robot robot-solving process; therefore, I chose to design a semi-autonomous robot that could map out obstacles in its environment and plan a route around the obstacles. The robot uses an ultrasonic sensor to measure the distance of obstacles from itself and uses a stepper motor to turn the ultrasonic sensor to measure the distance of objects from itself that aren't right in front of the robot. The robot uses two dual H bridge circuits, one to control the stepper motor and another to control two DC motors that move the robot. The brain of the robot is an Arduino Nano (microcontroller). Multiple parts of the robot are 3D printed: the chassis, the blinders for the ultrasonic sensor to focus its range, and the mount for the ultrasonic sensor. The ultrasonic sensor mount was designed to keep the measuring point of the sensor in the center of the robot and not travel in an arc when the stepper motor is turning.



scaner_4 §

```
long duration;
long distance;
float cardist;
float x;
float y;
int a[4] = {1,2,3,4};
int n;
int i;

void setup() {
  Serial.begin(9600);
  pinMode(2,OUTPUT);  //stepper
  pinMode(3,OUTPUT);  //stepper
  pinMode(4,OUTPUT);  //stepper
  pinMode(5,OUTPUT);  //stepper
  pinMode(6,OUTPUT);  //us sensor
  pinMode(7,INPUT);   //us sensor
  pinMode(10,OUTPUT); //motor 1
  pinMode(11,OUTPUT); //motor 1
  pinMode(12,OUTPUT); //motor 2
  pinMode(13,OUTPUT); //motor 2
  /*delay(5000);
  scan();
  delay(20000);*/
  movecar(1);
}

void stp() {
   digitalWrite(2,LOW);
   digitalWrite(3,HIGH);
   digitalWrite(4,LOW);
```