ME 102B: Final (Prototype) Manual

Uriel Barragan, Anjana Saravanan, Rohan Castelino

12 December 2021

Opportunity

The goal for this project was to create an electromechanical art piece which is both visually engaging and capable of producing interesting music.

Initial Strategy

To meet the outlined opportunity, we first decided that we wanted to create a stringed instrument (specifically a violin-like instrument) as there were few existing similar robots.

From there, we opted to create a relatively simple mechanical system in favor of adding complexity through software-synchronized replicas. This goal had the added benefit of ensuring that our project was more affordable and more likely to be completed within the limited time frame for the project.

Finally, we opted to use a crank-rocker and rack-pinion for the bowing and fretting mechanisms respectively with the actual violin being 3d printed from an open sourced design. Also, we planned to use a RaspberryPi 3 rather than an Arduino as it not only supports Python which aids in rapid coding/debugging but also had WiFi (e.g. wireless) capabilities

Key Considerations:

The reaction forces (F_r) of the pinion gears were calculated for each Pololu motor using the equations below. The max torque (τ_{max}) is taken from the spec sheet of each motor. The radius (r) of the pinion gear is 3.175 mm and both systems use the same rack and pinion. The results of the fretting and bowing systems are included in equations 3 - 6, respectively.

$$\Sigma\tau_a = I \cdot \underline{\alpha}_a = 0 \quad (1)$$

$$-\tau_{max} + F_x \cdot r = 0 \quad (2)$$

$$F_x \cdot r = \frac{\tau_{max}}{r} = 0.072\ kg\ \&\ 0.26\ kg \quad (3)$$

$$F_y = F_x \cdot tan(20°) = 0.025\ kg\ \&\ 0.096\ kg \quad (4)$$

$$F_r = \sqrt{F_x^{\,2} + F_y^{\,2}} = 0.077\ kg\ \&\ 0.28\ kg \quad (5)$$

$$\tau_{produced} = r \cdot F_r \cdot sin(20°) = 0.084\ kg{\cdot}cm\ \&\ 0.31\ kg{\cdot}cm \quad (6)$$

The Pololu motor from our lab kit was used to control the rack and pinion in the fretting mechanism. The motor has a 75.81:1 metal gearbox which allows for a stall torque of 1.3 kg·cm and an output torque of 0.23 kg·cm at 40% efficiency. Per the calculations above, the torque produced (0.084 kg·cm) is less than the stall torque (0.23 kg·cm), implying that the motor is powerful enough for the fretting mechanism. This motor also makes use of the encoders to allow for feedback control and ultimately allow the user to choose which note to produce.

A Pololu motor similar to the one in our lab kit was used for the bowing mechanism. The bowing mechanism consists of multiple components which results in a higher mass than our fretting mechanism. Thus, more torque is required which is why we chose to increase the gearbox ratio for the bowing motor. This motor has a 379.17:1 metal gearbox which allows for a higher stall torque of 5.5 kg·cm and an

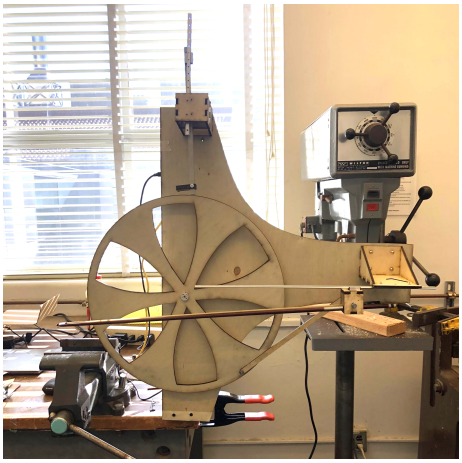
output torque of 0.84 kg·cm at 36% efficiency. The torque produced (0.31 kg·cm) is less than the stall torque (5.5 kg·cm) so the motor will work.

One of our main goals was to keep the design as cost efficient as possible so that it can be easily produced. Although bearings could have improved the mechanical performance of the systems, they are expensive and would make our cost efficient goal unattainable. Ideally, we would be able to produce four of these violin setups and have them play synchronously.

Results:

Mechanical Design:

As discussed earlier, we originally wanted to use a crank-slider for the bowing as it was aesthetically engaging. However, during testing, it was found that at key points, the vertical force on the slider led the mechanism to jam. Due to time constraints, we decided to implement a rack and pinion for the bowing mechanism, while also adding a second mount to increase the stability of the bow rail. The bowing motor was also repositioned in order to control the bow rack.

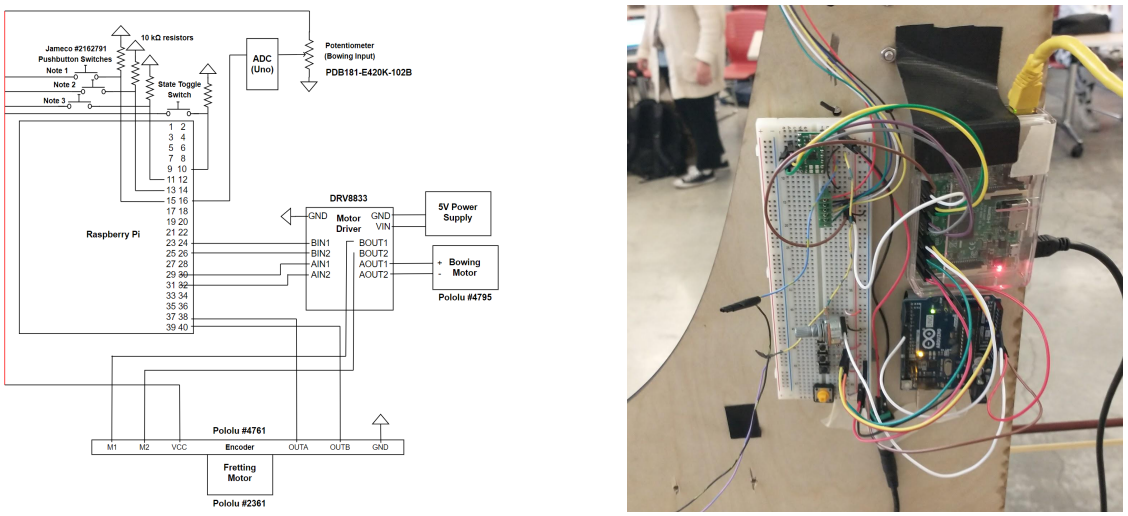


(a)

(b)

Figures 1a-b. The initial slider-crank and the final rack-pinion assemblies

Circuit Design:



(a) (b)

Figure 2a-b. Circuit diagram and final circuit with fretting motor, analog/digital inputs, and Raspberry Pi/Arduino

Software Design:

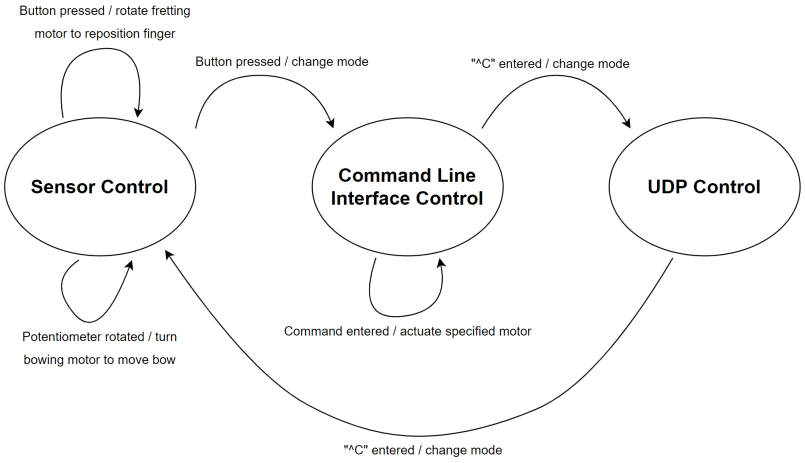


Figure 3. Implemented state machine

Discussion:

After wrapping up the prototype, there is a long list of changes that could be made to improve its performance. In looking at this list, there are three key lessons to be learned to avoid similar problems in future projects.

1. Mechanical prototyping is going to be a bottleneck for the project and should be first prioritised over software/electrical work. Creating a clean CAD model and manufacturing parts will not only take longer than expected but also has more unavoidable hurdles (e.g. limited machine shop hours and lead times on ordered parts). Finally, the control scheme can't be tuned easily until the mechanical system is done.
2. Additionally, the mechanical design should be adjustable to allow for manufacturing tolerances and unexpected design flaws. We ran into issues with getting the right spacing between

rack/pinion gears as well as the right angle between the fretting finger and violin neck which would have required remanufacturing to fix/maintain due to the rigid design.

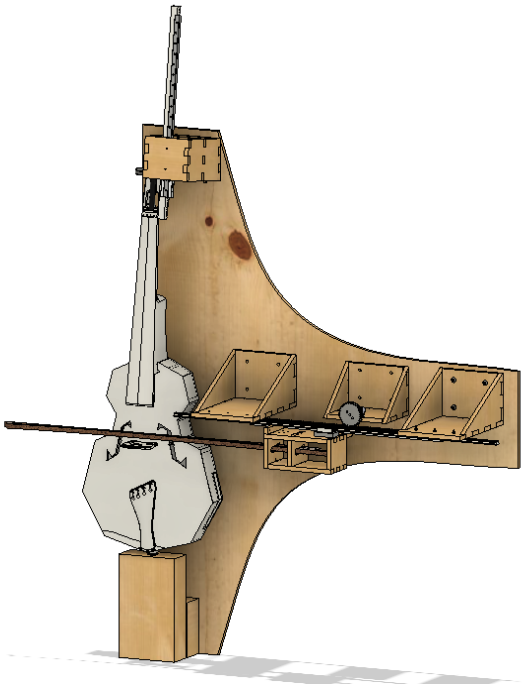
3. Finally, while starting earlier and allowing for adjustments can help a lot, sometimes core components (e.g. the crank-slider) will just not work and those central flaws need to be discovered early on rather than only when the full product is assembled. Instead, the design should be broken into simplified, subcomponents which can be separately tested easily. For instance, this philosophy was taken into approach with the code which had a modular design and where non-critical features/minor bugs were pushed off to be implemented down the line.

Appendix A: Bill of Materials

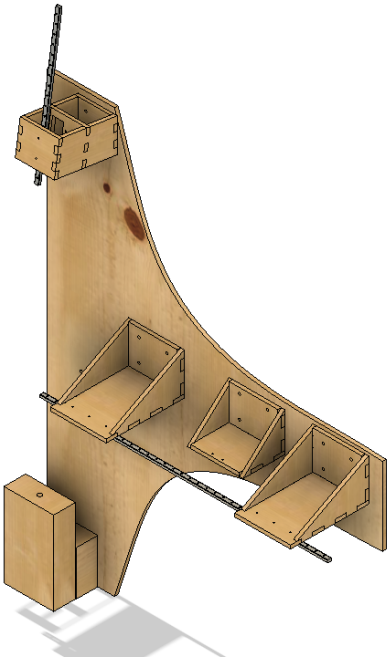
Mechanism	Part	Description	Price	Qty	Source	Source Link
Bowing Mechanism	Rail	515mm length ($0.17/mm) Part Diagram	$90.31	1	McMaster	9829K114
	Violin bow		$22.99	1	Old Town School of Folk Music	Palatino Violin Bow
	Carriage	Part Diagram	$21.91	2	McMaster (1)/Lasercutting (1)	9829K1
	Rail-Base Screws	M2 x 12mm, 25 pack	$7.44	4	(Already have on hand)	92095A455
	Bow Grip-Bow Screw	M3 x 30mm, 50 pack	$6.43	2	McMaster	92095A187
	Bow Grip-Carriage Screw	M2 x 20mm, 25 pack	$6.09	4	(Already have on hand)	92095A106
	Rail-Base Nuts	M2, 100 pack	$2.06	4	(Already have on hand)	90592A075
	Bow Grip-Bow Nut	M3, 100 pack	$1.17	2	(Already have on hand)	90592A085
	Base		$0.00	1	Lasercutting	N/A
	Bow Grip		$0.00	1	Lasercutting	N/A
	Rack		$0.00	2	Lasercutting	2662N57
	Pinion gear		$0.00	1	Lasercutting	2662N13
	Pinion gear shaft hub	Mounting hub	$0.00	1	Lab Kit	1078
	Motor	380:1 Micro Metal Gearmotor HP 6V	$15.95	1	Pololu	4794
	Motor Base		$0.00	1	Lasercutting	N/A
	Motor Mount	Micro motor mount	$0.00	1	Lab Kit	989
	Rail Support Base		$0.00	2	Lasercutting	N/A
Fretting Mechanism	Rail		$33.93	1	McMaster	9728K6
	Carriage	13" ($2.61/1")	$29.94	1	McMaster	9728K31
	Pinion gear		$9.00	1	McMaster/Lasercutting	2662N13
	Rack		$4.91	1	McMaster	2662N57
	Base		$0.00	1	Lasercutting	N/A
	Finger		$0.00	1	Lasercutting	N/A
	Pinion gear shaft hub	Mounting hub	$0.00	1	Lab Kit	1078
	Motor	75:1 Micro Metal Gearmotor HP 6V	$0.00	1	Lab Kit	2361
	Motor Mount	Micro motor mount	$0.00	1	Lab Kit	989
	Encoder	Encoder for Micro Motors	$0.00	1	Lab Kit	4760
Violin Body	Tuners		$27.52	1	Amazon	Grover 9NB
	Strings		$21.99	1	Old Town School of Folk Music	Prelude String Set
	Truss Rods		$14.99	2	Amazon	Carbon Fiber Tubes
	Tailgut		$3.95	1	Amazon	Tailgut
	Vbody		$0.00	1	3D Printing	N/A
	Neck		$0.00	1	3D Printing	N/A
	Pegbody		$0.00	1	3D Printing	N/A
	Tailpiece		$0.00	1	3D Printing	N/A
	Bridge		$0.00	1	3D Printing	N/A
		Total:	$320.58			
		Per team member	$106.86			

Figure 4. Screenshot of final bill of materials

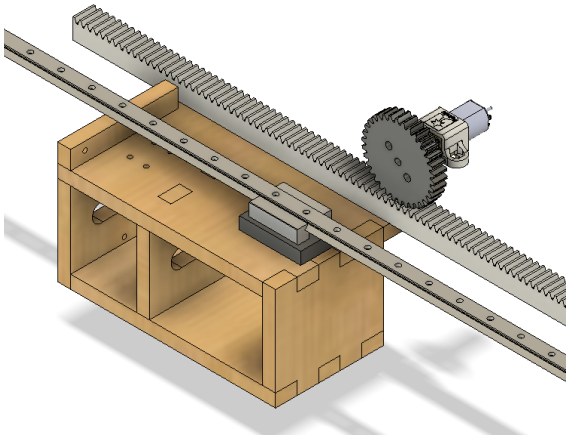
Appendix B: CAD



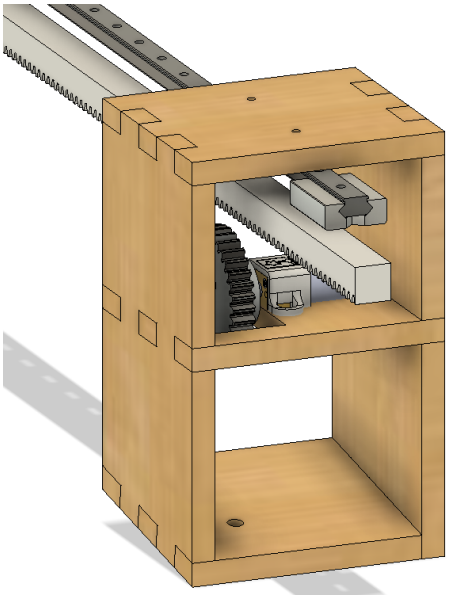
(a) Full assembly



(b) Base for attaching mechanisms



(c) Bowing mechanism



(d) Fretting mechanism

Figure 5. CAD of full assembly and subcomponents

Appendix C: Code

Note that code can also be found on GitHub here.

```
import signal
import socket
import argparse
import tty
import sys
import termios
import pdb


from gpiozero import Button, RotaryEncoder, Motor
import time, threading

# Custom argparse class so can catch errors cleanly
class ArgumentParserError(Exception): pass
class HelpRequestedError(Exception): pass
class ThrowingArgumentParser(argparse.ArgumentParser):
    def error(self, message):
        raise ArgumentParserError(message)
    def print_help(self):
        raise HelpRequestedError(self.format_help() + "\n **Replace bootstrap.py with
/lockbranch**")



class InputReceiver():
    def __init__(self) -> None:
        # State machine setup
        self.state_toggle = False
        self.state = "sensor_control"       # Initial case for when script first starts
        self.power_on = True                # Keep main loop running by default

        # UDP setup
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.server_socket.bind(('', 6667))
        self.server_socket.setblocking(0)

        self.error_Integral = 0;

        # CLI setup
        self.sentence = ""
        self.params = {"bow":"False","note":"0"}
        self.param_recast_dict = {"a":"0", "b":"1", "c":"2", "k":"5","on":True,
"off":False, }

        self.argparser = self.build_argparser()
        self.sentence = ""                  # Initializing sentence for CLI control
        # Interrupt handler for CLI
        def state_interrupt_handler(signum, frame):
            self.state_toggle = True
        signal.signal(signal.SIGINT, state_interrupt_handler)

        # Sensor setup
        self.GPIO_setup()

        self.frettingMotor = Motor("BOARD24", "BOARD26")
        self.bowingMotor = Motor("BOARD29", "BOARD31")
        self.frettingEncoder = RotaryEncoder("BOARD38", "BOARD40", wrap=False,
max_steps=10000)
        self.frettingEncoder.steps = 0
```

```
        self.desiredNote = 0
        self.currentNote = 0

        self.bowing = False
        self.bowingDirection = 1
        self.strokeTime = 0
        self.strokeStartTime = 0
        self.strokeTotalTime = 5 # 5 seconds per stroke?

        self.toggleNote = False # for testing


    def build_argparser(self):
        argparser =
ThrowingArgumentParser(formatter_class=argparse.ArgumentDefaultsHelpFormatter)
        argparser.add_argument("--bow", help="On or off?", choices=['on', 'off'])
        argparser.add_argument("--note", help="Select note to play", choices=
['a','b','c','k'])
        return argparser

    def process_sentence(self, sentence):
        new_params, unknowns = self.argparser.parse_known_args(sentence.split(' '))
        new_params = vars(new_params)

        for key in new_params:
            if new_params[key] == None:
                new_params[key] = self.params[key]
            else:
                new_params[key] = self.param_recast_dict[new_params[key]]
        self.params = new_params
        return new_params

    def stream_commands(self, params):
        print(f"Received {params}. Streaming now.")
        self.control_system(str(params))

    def GPIO_setup(self):
        def state_button_callback(channel):
            self.state_toggle = True

        def a_button_callback(channel):
            self.params['note'] = '0'
            self.stream_commands(self.params)

        def b_button_callback(channel):
            self.params['note'] = '1'
            self.stream_commands(self.params)

        def c_button_callback(channel):
            self.params['note'] = '2'
            self.stream_commands(self.params)

        def bow_input_callback_pressed(channel):
            self.params['bow'] = 'True'
            self.stream_commands(self.params)

        def bow_input_callback_released(channel):
            self.params['bow'] = 'False'
            self.stream_commands(self.params)

        # State toggle button
        self.state_button = Button("BOARD10", pull_up=False) # pulled low by default
        self.state_button.when_pressed = state_button_callback
```

```
        # Note A
        self.A_Button = Button("BOARD11", pull_up=False) # pulled low by default
        self.A_Button.when_pressed = a_button_callback

        # Note B
        self.B_Button = Button("BOARD13", pull_up=False) # pulled low by default
        self.B_Button.when_pressed = b_button_callback

        # Note C
        self.C_Button = Button("BOARD15", pull_up=False) # pulled low by default
        self.C_Button.when_pressed = c_button_callback

        # Bow toggle
        self.bow_input = Button("BOARD16", pull_up=False)
        self.bow_input.when_pressed = bow_input_callback_pressed
        self.bow_input.when_released = bow_input_callback_released

    def switch_state(self):
        self.state_toggle = False
        if self.state == "cli_control":
            # Reset CLI
            termios.tcsetattr(sys.stdin, termios.TCSADRAIN, self.orig_settings)
            # Set up GPIO
            self.GPIO_setup()
            # Switch state
            self.state = 'sensor_control'
        elif self.state == "sensor_control":
            # Reset GPIO pins
            self.state_button.close()
            self.A_Button.close()
            self.B_Button.close()
            self.C_Button.close()
            self.bow_input.close()
            # Save orig CLI and then set up custom CLI
            self.orig_settings = termios.tcgetattr(sys.stdin)
            tty.setcbreak(sys.stdin)
            # Switch state
            self.state = "udp_control"
        elif self.state == "udp_control":
            # Switch state
            self.state = "cli_control"
        print(f"--Switching state to {self.state}--")

    def delete_detected(self):
        x=sys.stdin.read(2)
        if x == "[3":
            x=sys.stdin.read(1)
            if x == "~":
                self.stream_commands(self.process_sentence("--note a"))
                self.power_on = False
                termios.tcsetattr(sys.stdin, termios.TCSADRAIN, self.orig_settings)
            else:
                print('ignoring special char')
        else:
           pass

    def get_text(self):
        char=sys.stdin.read(1)
        if char == chr(27):
            self.delete_detected()
        elif char == chr(10):
            print("")
            self.stream_commands(self.process_sentence(self.sentence))
            self.sentence = ""
        else:
```

```
            print(char,end="",flush=True)
            self.sentence+= char

    def main(self):
        print('--------State machine launched--------')
        print(f'--Switching state to {self.state}--')
        while self.power_on:
            # Match case to switch between states
            match self.state:
                case 'cli_control':
                    try:
                        self.get_text()
                        # Check for state switch
                        if self.state_toggle:
                            self.switch_state()
                    except ArgumentParserError as e:
                        self.sentence = ""
                        print(str(e))
                    #except:
                        #termios.tcsetattr(sys.stdin, termios.TCSADRAIN,
self.orig_settings)
                        #print('error')

                case 'sensor_control':
                    if self.state_toggle:
                        self.switch_state()

                case 'udp_control':
                    try:
                        dataFromClient, address = self.server_socket.recvfrom(256)
                        print(dataFromClient.decode())

self.stream_commands(self.process_sentence(dataFromClient.decode()))
                    except:
                        pass

                    if self.state_toggle:
                        self.switch_state()
        else:
                print('Powering off')
                self.server_socket.close()

    def bow_stroke(self):
        self.bowingDirection = not self.bowingDirection
        #print('thread started')
        t = threading.Timer(5, self.bow_stroke)

        if self.bowing:
            t.start()
            if self.bowingDirection == 1:
                print("bowing forwards")
                self.bowingMotor.forward(speed=0.25)
            else:
                print("bowing backwards")
                self.bowingMotor.backward(speed=0.25)
        else:
            self.bowingMotor.stop()
            t.cancel()

    def control_system(self, command):
        command = eval(command)

        # ----------- bow stuff -----------
        if isInstance(command['bow'],str):
            new_bow_command = eval(command['bow'])
```

```
        else:
            new_bow_command = command['bow']

        #print(new_bow_command)
        #pdb.set_trace()
        if (new_bow_command != self.bowing) & (new_bow_command == True):
            #print('if worked')
            self.bowing = True
            self.bow_stroke()

        elif new_bow_command == False:
            #print('stopping')
            self.bowing = False

        # ----------- note stuff-----------
        self.desiredNote = eval(command['note'])
        if self.desiredNote & self.desiredNote!= self.currentNote:

            note_increment = 220/4*2
            desiredNoteCount = self.desiredNote * note_increment

            currentNoteCount = self.frettingEncoder.steps

            error = desiredNoteCount - currentNoteCount
            while abs(error) != 0:
                if error > 0:
                    self.frettingMotor.forward(speed=0.4)
                else:
                    self.frettingMotor.backward(speed=0.2)
                currentNoteCount = self.frettingEncoder.steps
                error = desiredNoteCount - currentNoteCount
                print("Error: ", error) # debugging
                print("currentNoteCount: ", self.frettingEncoder.steps) # debugging


            self.frettingMotor.stop()
            self.currentNote = self.desiredNote
            print(self.frettingEncoder.steps)

if __name__ == "__main__":
    primaryInputReceiver = InputReceiver()
    primaryInputReceiver.main()
```