# FINAL REPORT ON THE MUSICIAN'S MATE: A ROBOTIC PAGE TURNER

**BY**

JEAN DE JESUS, SOPHY LI, HANNAH ROSENBERG, SHAFIN SHAHRIAR

20727757, 20703756, 20666114, 20707136

MTE 100 AND GENE 121

DECEMBER 4, 201

## Table of Contents

## List of Figures



## List of Tables

# 1 SUMMARY

In this report, the process carried out to design and implement the robotic page turner and volume data collection is described. The robotic arm was able to move the pages of sheet music on a music stand when cued by the musician. As the musician played, the robot collected volume data and outputted this data to a file to be analyzed by the user.

The robotic arm was built on a driving base that moved within a track and a claw was used to grasp the individual pages. The user cued the robot to turn a page by pressing on a foot pedal.

The code for the page flipping tasks, sound data collection tasks, and safety procedures was tested separately and then combined to be tested together.

The project was successful in meeting key constraints such as moving the pages for the musician and outputting the sound data to a file. However, it was unsuccessful in meeting some criteria such as being able to move pages in the reverse direction.

## 2 DESIGN PROBLEM DEFINITION

The problem that we attempted to solve was one that musicians encounter regularly during practice sessions. Many musicians have difficulties turning the pages of their music while they are practicing. Having to stop playing to turn a page creates an awkward, unwanted pause in the music. To resolve this need, we built a robot that moved pages for the musician when cued to do so.

There was also a secondary problem that we attempted to solve. When musicians are playing, it is difficult for them to determine for themselves at what volume they are playing and whether that volume matches the one that is indicated by the composer. To aid with this issue, our system outputted a file that displayed the volume ranges played by the musician over the course of the music.

# 3 CRITERIA AND CONSTRAINTS

## 3.1 Constraints

**Table 3.1 Design Constraints of System**

| Constraint | Measurement of Success |
|---|---|
| Grabs page without damaging it | -Arm is able to grasp the page without ripping it<br>-Determined through trial and observation |
| Moves page in timely manner | -Arm is able to move page within 2-3 seconds<br>-Determined with timer |
| Moves page on cue | -Arm begins to move when user hits foot pedal<br>-Determined through trial and observation |
| Arm is stable | -Weight of robot is distributed equally so that it remains upright<br>-Determined through trial and observation |
| Releases page in correct location | -Releases page only when it is no longer covering next page<br>-Determined with ruler, approximately 22 cm from start location |
| Successful pause procedure | -During a page flip, if the user holds the centre button, the robot will stop, open the claw, and pause for 5 seconds, then continue<br>-Determined through trial and observation |
| Successful shutdown procedure | -If user holds the centre button not during a page flip, the program will end<br>-Determined through trial and observation |
| Detects volume and successfully outputs to file | -Outputs every 2 seconds (or 7 seconds if a page turn has occurred) the average volume over the course of a second<br>-Verified with output times listed in file |
| Outputs volume ranges to file accurately | -Successful if volume range outputted generally matches relative volume played by musician<br>-Determined through observation |

Throughout the project, the constraints, shown in Table 3.1, provided a clear outline for the goals we were attempting to achieve. However, some of the constraints were more useful than others. The most important constraints were that the robot could move pages on cue and release a page in the correct location. Without meeting these two conditions, our system would not have been of any use

to a musician. We were constantly readjusting our mechanical and software design to ensure that these constraints were met.

On the other hand, constraints such as the arm being stable and moving pages without damaging them were not entirely necessary. In the case of the arm being stable, the robot would not have been able to meet the other constraints, such as moving pages on cue, without this stability, and, therefore, this constraint was redundant. As for moving the pages without damaging them, this condition was also already implied since the musician could not use the page if it was damaged.

Being able to detect volume and successfully output it to a file was another important constraint because without this area being successful, the entire portion of our project involving sound sensing data collection would also not have been successful. We would not have been able to meet all of the project requirements involving the use of different types of sensors without successfully outputting sound measurements.

Finally, having a successfully pause procedure and shutdown procedure was critical. These constraints ensured that the system was safe to use.

A constraint that could have been added would have involved the robot being able to autonomously move to its start location. This was something that we implemented regardless of the fact that it was not one of our constraints. It was an important part of the functioning of our system because the user did not have to place the robot in the perfect location at the start for it to function.

## 3.2 Criteria

**Table 3.2 Design Criteria of System**

| Criteria | Measurement of Success of Criteria | Benefit to Project |
|---|---|---|
| Portability and Weight | Lighter and more compact system (i.e. lower volume and surface area taken up) is better | More convenient for musicians travelling back and forth with music |
| Adjustability of height of system | If system will function at a variety of different heights | Allows user more flexibility in music stand height according to preference |
| Ability to move pages in both directions | If robot can move page back to original location when cued by user | Lessens human interference and added convenience for user |
| Moves pages smoothly | Moves page in one motion without sudden stops or increases/decreases in speed | Less distracting to musician while playing |

Regarding the criteria, shown in Table 3.2, we achieved significant success in the adjustability of the height of the system as well as moving pages smoothly. These criteria contributed to the usability of the system in different environmental conditions and prevented the user from being distracted by it. We were unable to make significant progress in the portability section as our design involved a mechanism that prevented the ability to collapse the music stand. As for the ability to move pages in both directions, this was sacrificed in order for us to properly achieve the ability to flip pages forward smoothly and have them released in the correct location.

A criterion that would have been worth adding was durability. Our final project functioned well, however, it may not have continued to function for an extended period of time. We used materials such as paper and cardboard where other, longer-lasting materials could have been beneficial. Also, our design relied on the elasticity of plastic tabs, causing them to pop up when a page was turned. This may have worn out over time.

# 4 MECHANICAL DESIGN AND IMPLEMENTATION

## 4.1 Robot Base and Arm Design

The robotic arm and base incorporated three motors: two continuous rotation motors at the base of the robot that controlled leftward and rightward movement in the track and one standard servo motor that opened and closed the gripper to grab the pages. We originally debated between building a stationary, pivoting mechanical arm and an arm mounted on a driving base. We found that the driving base (Figure 4.1) gave us more flexibility. One aspect of our design involved a pause function that could be initiated by the user during a page turn. It stopped the robot, released the page, waited five seconds, and then continued to move the page. Had we chosen a pivoting arm design, the gripper would not have been in the correct location to grab the page again mid-turn as the pivot would have taken a radial path. As well, the pivoting arm system would have been less autonomous as it would have required the user to place the base in the exact correct location. The driving robot could be placed anywhere in the track and move autonomously to its start location. The tradeoff that we made when choosing the driving arm option was less portability. The driving robot required a track to ensure it remained parallel with the music stand. This meant the track had to be mounted on the music stand, making it no longer collapsible.

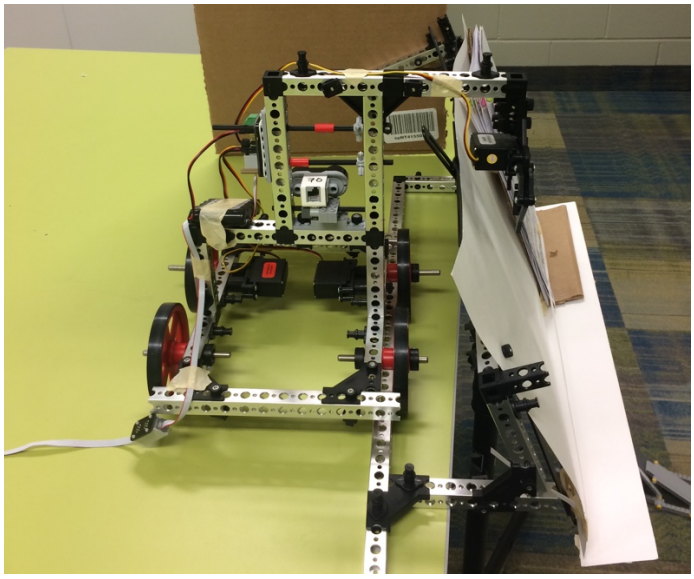


**Figure 4.1 Robotic arm and base in track**

One possible alternate design for the track was to use a 3D printed dowel. This dowel would have been passed through the holes of the Tetrix base and mounted to the music stand. In the end, we opted to build the Tetrix track over the 3D printed dowel track, as shown in Figure 4.1. We found that the Tetrix track was successful in keeping the robot parallel even though it only created a barrier on one side of the wheels. We decided against the dowel design as it would have been much more difficult to mount on the music stand in a stable fashion and would have been made of a much less durable material than the Tetrix pieces.

## 4.2 Sensor Attachment Design

The robot incorporated three sensors: a touch sensor, an ultrasonic sensor and an NXT sound sensor. The touch sensor, used to activate the page-turning mechanism, was placed on the floor close to the musician's foot. We used a design in which the user hit a foot pedal that activated another intermediary pedal before hitting the touch sensor (Figure 4.2). This was preferable to a pedal that directly hit the touch sensor as the intermediary pedal helped to prevent any damage to the touch sensor. As well, the second pedal helped to position the mechanism at an angle that was more comfortable for the user.

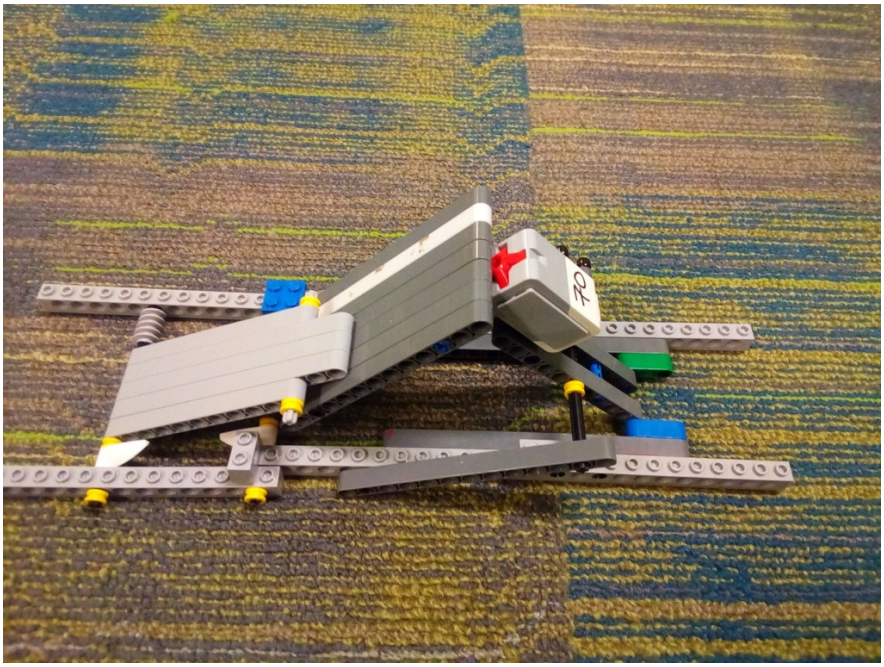


**Figure 4.2 Foot pedal design**

It was also necessary to design a sensor system that would allow to robot to start and stop at the correct positions. Originally, we intended to mount a touch sensor at the end of the track so that the robot would stop when it hit the touch sensor. Then, to return to its original location, it would use the motor encoder value of the distance it took to drive to the end of the track. However, this system would not have allowed for the robot to autonomously move to the correct start location without being placed there properly in the first place. Instead, we used an ultrasonic sensor that took distance readings from a wall placed at the beginning of the track. This design allowed for the robot to autonomously drive to its start location at the beginning of the program. One of the tradeoffs of this design was that it decreased the portability of the system as we had to attach a wall, as shown in Figure 4.1, to the end of the music stand. Despite this, the autonomous abilities of the robot were worth the decrease in portability.

The last sensor that the system incorporated was an NXT sound sensor which recorded volume readings. This was attached to the main brick, and placed half of a metre away from the user.

## 4.3 Gripping and Turning Pages

For the system to turn pages effectively, the mechanical design had to allow for the robot to accurately find the location of the paper, grab the paper without damaging it, and move it to the correct new location. Accordingly, we chose to use the Tetrix claw to grasp the pages. However, inserting the claw between pages would have been a difficult task. Instead, we chose to position the claw parallel to the page and grabbed tabs protruding approximately 3 cm perpendicular to the sheet of music as show in Figure 4.3. Each tab was mounted at the same spot on each sheet of music, and the starting sheets of music were stacked one on top of the other. The tradeoff of this design was that the system could not function for all sheet music, only for unbound A size paper with these specific tabs attached to them. Although this may be true, unbound paper is most commonly used by orchestra musicians and the tabs did not obstruct the user's view of the music.

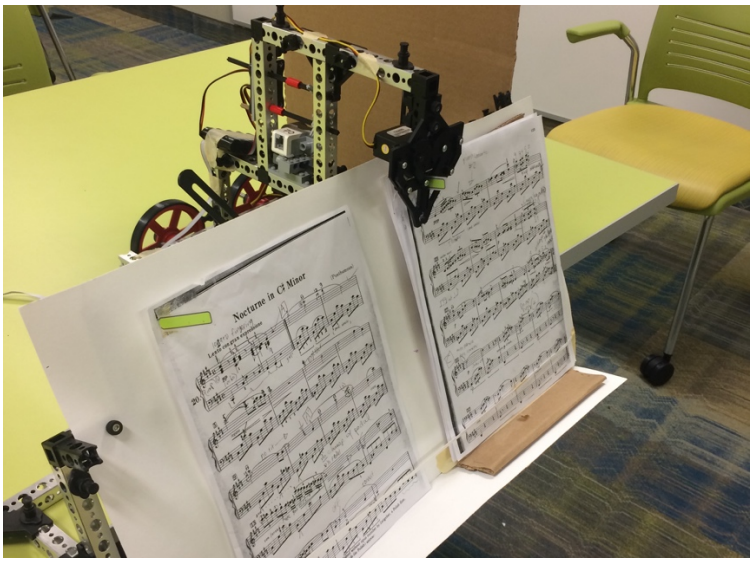



**Figure 4.3 Tabs attached to sheet music**

Originally, we created these tabs using paper. However, the robot could only successfully grab about three pages in a row on average. To improve consistency, the material of the tabs was modified to be plastic, as shown in Figure 4.3. The plastic popped out of the page more easily so that the gripper would successfully grab the tabs more consistently. This change increased the number of pages that the robot could turn to four.

Another problem we encountered was that after a few sheets of paper were moved, the pile of used sheets was thicker than the pile of new sheets. So, when the robot was moving the next page, it was inserting it between the used sheets instead of on top of them. To solve this problem, we mounted cardboard (Figure 4.4) behind and below the new sheets of music so that this pile would protrude more than the pile of old sheets. The new adjustment made it almost impossible, in the timeframe that was left, for us to design a function in which the robot moved a page back to its original location. This is because the cardboard was blocking the path. We determined that the benefit gained from this tradeoff was worthwhile since orchestra musicians could have to play very long pieces and the ability to turn more pages forward was more useful than turning pages backwards.

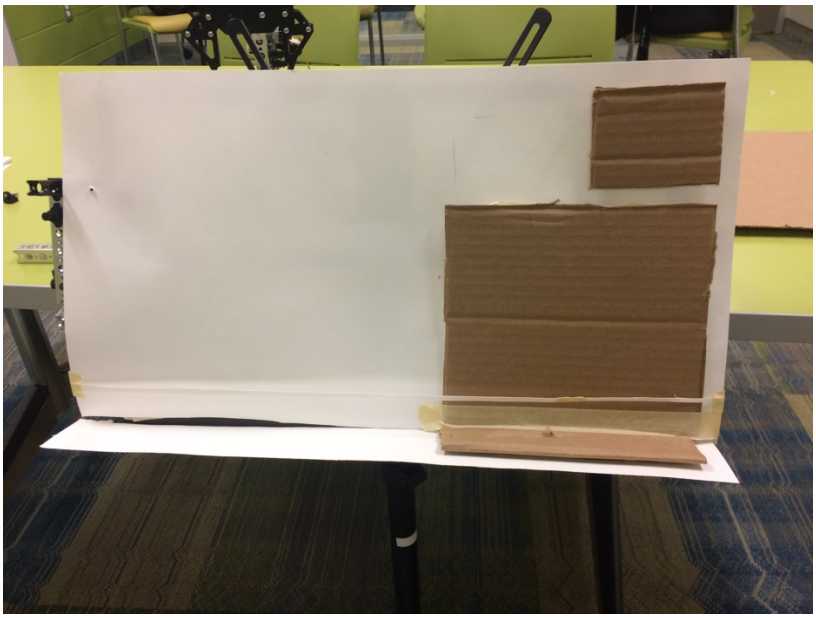


**Figure 4.4 Configuration of music stand**

Lastly, we attached a thin barrier made out of poster paper onto either side of the music stand (Figure 4.4). This prevented the pages from falling off while still being thin enough and low down enough that it did not obstruct the user's view of the music. With these new design solutions, the robot could consistently turn eight pages.

## 4.4 Overall Assembly

The only major issue that we encountered with overall assembly was organizing the wiring. Since the different components of the system were very spread out, some thought had to go into arranging them. Seeing as the wall was on the right side of the track, we placed the EV3 brick on the left side. We mounted the sound sensor directly onto the EV3 brick and angled it toward the user. To ensure that the foot pedal would reach the ground, it was also placed on the left side of the stand. This was the configuration that prevented the wires from getting caught on stationary parts as the robot moved.

# 5 SOFTWARE DESIGN AND IMPLEMENTATION

The program was designed to accomplish three main tasks: to flip an individual page every time the user pressed on the foot pedal, to continuously take volume measurements and output them to a file throughout the program, and to be able to pause the program as a safety measure.

## 5.1 Functions Used and Their Justifications

The Flip Page function was used repeatedly throughout the program since it would occur every time the user hit the foot pedal. As such, it was appropriate to choose this as a function. Had we left all of this code in the main function the code would have been far more disorganized and difficult to read.

Since the robot repeatedly moved back and forth in many different locations throughout the program, creating one Robot Drive function that was capable of powering the robot at different speeds was beneficial. This made the code more readable and efficient. All of the code for powering Tetrix motors using RobotC made use of the EV3Servo-lib-UW.c library along with instructions provided by the teaching team [1].

Taking volume measurements and outputting them to a file was also something that occurred continuously throughout the program. The Measure Volume function called the other two functions, Collect Readings and Output Volume inside of it. These two tasks were separated into two functions for better readability and organization. Although they were all part of the measure volume system, they each served a different purpose within the system. Had all three functions been combined into one it would have been unreasonably long. As well, having them separate made the debugging and testing stage much easier to carry out. Having the Measure Volume Function call up Collect Readings and Output Volume rather than Main also contributed to readability and organization. The code for the file output using RobotC made use of the EV3_FileIO.c library along with instructions provided by the teaching team [2].

The pause function was included as a separate function because it was called in various instances in the program. Within the Flip Page function, it was called after the claw closed, when the robot was moving forwards, and when it was moving backwards. Since it was used on multiple occasions, it increased efficiency and readability to make it a separate function.

## 5.2 Testing Throughout Project

Various stubs were used throughout the project to test whether individual functions executed their respective tasks. Then, these functions were combined into subsystems: page flipping, sound data collection and output, and safety procedures. These subsystems were then tested to determine if they worked together collaboratively as intended.

Various calibrations and tests were also conducted to determine certain constant values, such as the amount of time to wait in the pause function, the range of ultrasonic readings from the beginning to the end of the track, and the volume ranges measured by the sound sensor. These values were determined experimentally using stubs. As well, we used the Demo Function provided by the Teaching Team [3] to determine the calibration of the Servo motors from the Tetrix kit. We then offset the motor speed from the deadband to a power of 25. The proper motor power was determined based on the smoothness of the robot's movement.

When testing our individual functions, we first tested the Robot Drive function since it was called up by the Flip Page function. Once the robot was successfully moving in a straight line along the track, we proceeded to test the Flip Page function. Then, after fixing all logic errors, the robot could successfully grab pages, move them to the end of the track, and return to its original location.

After testing the functions related to flipping pages, we separately tested the volume sensing and outputting functions. First, we tested the Collect Readings function by outputting its values to the EV3 display. Once those were deemed

satisfactory, we were able to test the Measure Volume functions and Output Volume functions. Once these functions could successfully output the volume ranges and times to the file, we were able to move on.

Our last task to test was the pause function. According to our software design, this function was supposed to be initiated when the user pressed the centre button. So, we added checks in the code to see if the user hit the centre button after the claw closed, when the robot moved forwards in the track, and backwards in the track. Then, we tested if the function successfully paused in each of those instances, and was able to continue afterwards with the task it was performing before the pause was initiated.

To conclude our tests, we did a general test of our main function. This involved ensuring that all of the proper messages to the user were being displayed on the screen and that the robot could successfully move to its correct start location at the beginning of the program. We then checked to see if the robot could successfully perform all of the tasks, the page flipping, the volume sensing, the file output, and the pause function, in one run of the program. Lastly, we verified that the entire program would shut down when the centre button was pressed (not during a page flip).

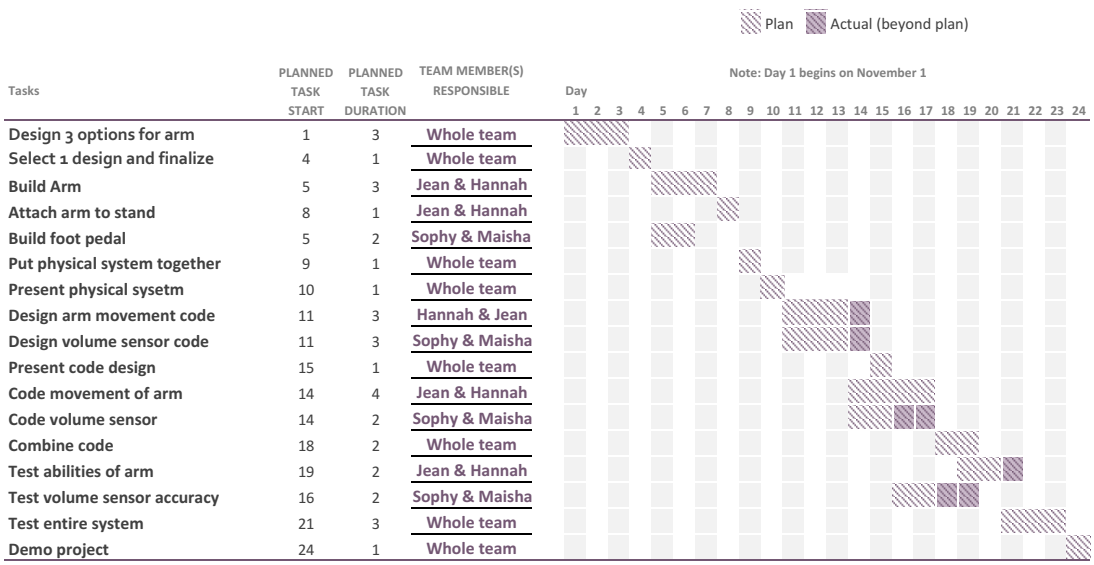
## 5.3 Redesign Based on Problems Experienced

Originally, the sound measurements were categorized into four ranges: piano, mezzo piano, mezzo forte, and forte. After testing the code along with the mechanical system, we determined that the sound sensor could not properly distinguish between the various volume ranges when music was played as the ranges were too small. To increase the accuracy of information outputted to the file, three sound categories were used instead: piano, mezzo forte and forte.

Another problem that we experienced was in our pause procedure. When a pause was initiated by the user in the middle of a page flip, even though the claw would close again before continuing, it could not successfully grab the page again. This was because when the page was released, it fell backwards a few centimetres since it was being pulled at a slight angle. To resolve this issue, we coded the robot to back up a small amount after the pause was over. We experimentally quantified this small amount and found that this solution was effective in resolving this problem approximately nine times out of ten.

In the initial drafts of the code, there was also a metronome component included that was designed to run continuously from the time that the user started the program to when it ended. However, there were difficulties implementing this within the overall structure of the code. We were unable to code a ticking sound on a continuous loop since there were other tasks being carried out at the same time. For instance, the page turning function took about seven seconds to complete. This meant that whenever a page was being turned, the metronome could not also be ticking at 60 or more beats per minute. We also experienced this issue with the volume sensor. We initially wanted to output the volume measurements every two seconds. To work around this issue, we simply waited seven seconds to output the volume when there was a page turn and outputted a time along with each volume measurement. However, we could not use this solution for the metronome as there could not be seconds in which there were no ticks. Since we were unable to brainstorm a solution even after much effort was invested, we concluded that it would be preferable to remove the metronome component. This allowed us focus on the fundamental aspects of the program, such as the page flipping and the sound measurements.

# 6 PROJECT MANAGEMENT




| Tasks | PLANNED TASK START | PLANNED TASK DURATION | TEAM MEMBER(S) RESPONSIBLE |
|---|---|---|---|
| Design 3 options for arm | 1 | 3 | Whole team |
| Select 1 design and finalize | 4 | 1 | Whole team |
| Build Arm | 5 | 3 | Jean & Hannah |
| Attach arm to stand | 8 | 1 | Jean & Hannah |
| Build foot pedal | 5 | 2 | Sophy & Maisha |
| Put physical system together | 9 | 1 | Whole team |
| Present physical sysetm | 10 | 1 | Whole team |
| Design arm movement code | 11 | 3 | Hannah & Jean |
| Design volume sensor code | 11 | 3 | Sophy & Maisha |
| Present code design | 15 | 1 | Whole team |
| Code movement of arm | 14 | 4 | Jean & Hannah |
| Code volume sensor | 14 | 2 | Sophy & Maisha |
| Combine code | 18 | 2 | Whole team |
| Test abilities of arm | 19 | 2 | Jean & Hannah |
| Test volume sensor accuracy | 16 | 2 | Sophy & Maisha |
| Test entire system | 21 | 3 | Whole team |
| Demo project | 24 | 1 | Whole team |

**Figure 6.1 Gantt chart for completed group project schedule**

As a team, we made the effort to split the tasks equally based on each individual's strengths. During the mechanical design phase, Jean and Hannah predominantly worked on building the arm while Sophy and Maisha worked to build the foot pedal. Then, during the software design and coding phase, Hannah and Maisha worked on putting the different pieces of code together while Jean and Sophy worked out the problems in the mechanical design. All of this occurred with much communication between the software side and the mechanical side so that changes in one would be reflected in the other.

As a whole, we were able to manage our time to ensure that we stayed relatively on schedule, as can be seen in Figure 6.1. There were occasions when we went over the allotted time for a task, however, this was never enough to cause our whole project to be behind. We left enough buffer time in our original schedule to allow for scenarios in which we fell behind. This way, we were able to be prepared and have our project ready on time.

# 7 CONCLUSIONS AND RECOMMENDATIONS

## 7.1 Conclusions

To summarize, the process of designing our system involved many different stages including planning the constraints and criteria, designing and building a physical system, designing and coding the software, and finally, testing the whole system. Overall, we met all of our milestones and remained relatively on schedule to complete the final product.

Overall our project succeeded in achieving the majority of the goals that we set for it. It was able to turn up to eight pages for a musician in a timely manner. It also provided the user with useful volume data that could contribute to the success of a practice session. However, it was not successful in meeting some of our criteria. It was not very portable, it could not flip pages in the reverse direction, and the motion of the page flipping was somewhat smooth although not entirely. As well, had we had more time and experience, we could potentially have resolved the problems we encountered with the metronome design and code.

## 7.2 Recommendations

Of the functions that we wrote, one that could have been refined was the Flip Page function. The aspect of this function that could have been improved upon was its length. For a function, it was quite long and it would have been preferable to divide it into a couple of smaller functions. We could have had one function for grasping and moving a page and another function for moving the robot back to its start location. Additionally, had we had more time and a higher skill level, we could have attempted to increase the autonomous abilities of the robot. Instead of the user having to cue the robot to turn the pages, we could potentially have had the robot read the music and determine itself when to flip a page.

As well, we could have designed a much more useful pause function. One design decision that could have improved this function would have been to give the user more control. Instead of hard-coding the length of the pause, we could have simply waited until the user pressed a button to cue the robot to continue again.

The most significant improvement that could have been made to the project would have been to implement a function that would flip the pages back when cued by the user. This would have been particularly useful to a musician, especially in a case where there was a repeat in the music. This occurs quite often in music and requires the user to go back to an earlier spot in the piece. To implement this, it would have required a significant redesign of our mechanical system. As mentioned previously, in order to properly flip pages forward and have them remain in the correct order, we chose to elevate the original pile of pages and have them protrude from the music stand so that they would land on the top of the second pile. However, to resolve this, we could have, instead, attached another motor to the music stand. This motor could have controlled an arm which would compress the new pile of pages, ensuring that the page being moved would land on top of the pile of old pages. Then, all of the sheets would have remained at the same level and we could have implemented a Flip Back function.

Another improvement that could have been made to the project would have been to make the metronome. Mechanically speaking, this would not have been exceedingly difficult. We had designed a prototype that, with some testing and adjustments, likely would have functioned. However, implementing the code was going to be very difficult. One potential solution could have been to play a different sound file for each metronome speed. Although, this would have been very tedious as it would have required the use of upwards of 30 sound files. It would have contributed significantly less to the complexity of the project than the physical metronome would have. The ideal solution would have required us to learn how to code multitasking in RobotC which we did not feel was a reasonable endeavour at the time. If the project were to have been extended, resolving this issue would have been a valuable undertaking.

To conclude, although our system can generally be considered a success, there will always be improvements that can be made.

# 8 REFERENCES

[1] M. Stachowsky and D. Lau, *Tetrix Motor Demo Program Written in RobotC to Run on the Lego EV3,* 2017.

[2] Teaching Team, *Using Files on the Lego EV3 and NXT Robots with RobotC,* 2017.

[3] M. Stachowsky and D. Lau, *Using the Tetrix Prime Motors with RobotC and a Legot EV3 Brick,* 2017.

# 9 APPENDIX A

```
#include "EV3Servo-lib-UW.c"
#include "EV3_FileIO.c"
//These libraries were provided by the teaching team

const int STOP_MOVING = -12;
const int FORWARDS = 25;
const int BACKWARDS = -25;
const int MOTORG = 1;
const int MOTORY = 2;
const int GRIPPER = 4;
const int TRACK_END = 42;
const int TRACK_BEG = 18;

/*
Assigns a power to the motors. If the robot is not stopping, it accounts for
deadband to ensure the two motors are moving at the same speed. The motors
were also mechanically positioned in opposite directions so they had to receive
opposite power values.
Coded by Shafin Shahriar
*/
void robotDrive(int motorPow)
{
	if (motorPow == STOP_MOVING)
	{
		setServoSpeed(S1, MOTORG, motorPow);
		setServoSpeed(S1, MOTORY, motorPow);
	}
	else
	{
		setServoSpeed(S1, MOTORG, motorPow, -3, -17);
		setServoSpeed(S1, MOTORY, -motorPow, -11, -22);
	}
```

```
}

/*
Stops robot, opens claw, and waits 5 seconds
Coded by Jean De Jesus
*/
void pause ()
{
	robotDrive (STOP_MOVING);
	setGripperPosition(S1, GRIPPER, 70);
	wait1Msec(5000);
}

/*
This function grabs a page, moves the page to the end location, releases the
page, and returns to its start location. It also has checks to see if the user
has hit the pause button throughout. If the user chooses to pause the function
while it is flipping a page, the claw grasps the page again afterwards before
continuing.
Coded by Hannah Rosenberg
*/
void flipPage ()
{
	setGripperPosition(S1, GRIPPER, 70);

	wait1Msec(500);

	setGripperPosition(S1, GRIPPER, 0);

	wait1Msec(500);

	if (getButtonPress(buttonEnter))
```

```
    {
          pause();
    }

      robotDrive (FORWARDS);

          while (SensorValue[S3] <= TRACK_END)
          {
                if (getButtonPress(buttonEnter))
                {
                      pause();
                      robotDrive(BACKWARDS);
                      wait1Msec(50);
                      robotDrive(STOP_MOVING);
                      setGripperPosition (S1, GRIPPER, 0);
                      wait1Msec(500);
                      robotDrive(FORWARDS);
                }
          }

          robotDrive(STOP_MOVING);

          setGripperPosition(S1, GRIPPER, 70);

          wait1Msec(500);

  robotDrive (BACKWARDS);

          while (SensorValue[S3] >= TRACK_BEG)
          {
                if (getButtonPress(buttonEnter))
                {
                      pause();
```

```
                        robotDrive (BACKWARDS);
                    }
                }

        robotDrive(STOP_MOVING);

}

/*
This function first calculates the average volume played over the course of a second by dividng the sum of the readings by the number of readings taken. It then checks if the average volume fits into the piano, mezzo forte, or forte range (or if no sound was made). It then outputs the range and the time the range was played at to a file.
Coded by Sophy Li
*/
void outputVolume(int numReadings, int sum, TFileHandle & fout)
{
        float volume = 0;
        volume = (float)sum/(float)numReadings;
        string volRange = "Pause";

        if (volume <= 1)
        {
                volRange = "No sound";
        }
        else if(volume <= 15)
        {
                volRange = "Piano";
        }

        else if(volume <=55)
        {
```

```
		volRange = "Mezzo Forte";
	}

	else
	{
		volRange = "Forte";
	}

	writeLongPC(fout, time1[T3]/1000);
	writeTextPC(fout, volRange);
	writeEndlPC(fout);
}

/*
This function takes sound readings continuously for a second and adds them
together. It also counts the number of readings taken.
Coded by Jean De Jesus
*/
int collectReadings(int & numReadings)
{
	int sum = 0;
	time1[T2] = 0;
	while (time1[T2] <= 1000)
	{
		sum += SensorValue[S4];
		numReadings++;
	}

	return sum;
}

/*
This function finds the sum of all the readings taken in one second and the
```

```
number of readings taken using the Collect Readings function. It then passes
them to the Output Volume function.
Coded by Sophy Li
*/
void measureVolume(TFileHandle & fout)
{
	int numReadings = 0;
	int sum = 0;

	sum = collectReadings(numReadings);
	outputVolume(numReadings, sum, fout);
}

//Coded by Hannah Rosenberg
task main()
{
	SensorType[S1] = sensorI2CCustom9V;
	SensorType[S2] = sensorEV3_Touch;
	SensorType[S3] = sensorEV3_Ultrasonic;
	SensorType[S4] = sensorSoundDBA;

	TFileHandle fout;
	bool fileOkay = openWritePC(fout, "soundReadings.txt");

	robotDrive (STOP_MOVING);

	displayString (0,"Push centre button to start");

	while (!getButtonPress(buttonEnter))
	{}

	while (getButtonPress(buttonEnter))
	{}
```

```
	eraseDisplay();

	displayString (0,"Hold centre button during");
	displayString (1,"page flip to pause");
	displayString (3,"Else, hold centre button");
	displayString (4,"to end program");

//moves robot to correct start location

	if (SensorValue[S3] > TRACK_BEG)
	{

		robotDrive(BACKWARDS);

		while (SensorValue[S3] > TRACK_BEG)
		{}

		robotDrive (STOP_MOVING);
	}

	time1[T1] = 0;

	time1[T3] = 0;

/*
Continuously checks to see if user has hit foot pedal or if 1 second has
passed. If the user has hit the foot pedal, the Flip Page function is called.
If one second has passed, the Measure Volume function is called.
*/
	while (!getButtonPress(buttonEnter))
	{
		if (SensorValue[S2] != 0)
```

```
		{
			flipPage ();
		}

		if (time1[T1] >= 1000)
		{
			measureVolume(fout);
			clearTimer(T1);
		}

	}

	robotDrive(STOP_MOVING);

	writeTextPC(fout, "Duration of session: ");
	writeLongPC(fout, time1[T3]/1000);
	writeEndlPC(fout);

	closeFilePC(fout);
}
```

# 10 APPENDIX B

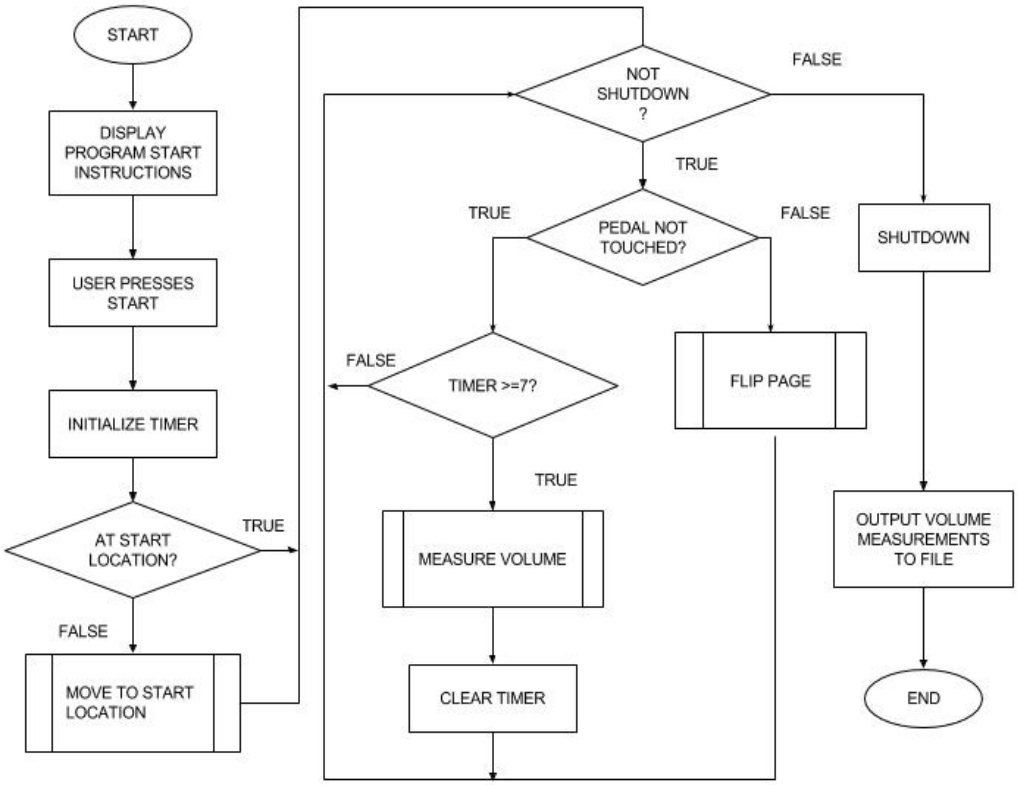



**Figure 10.1 Main function flow chart**

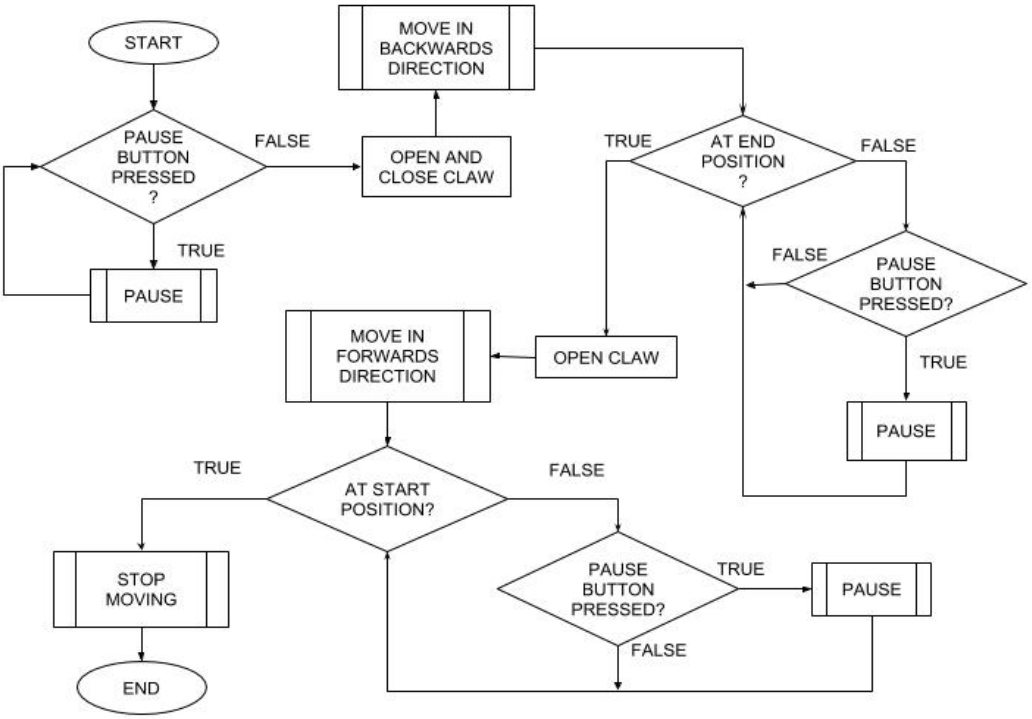



**Figure 10.2 Flip Page function flow chart**

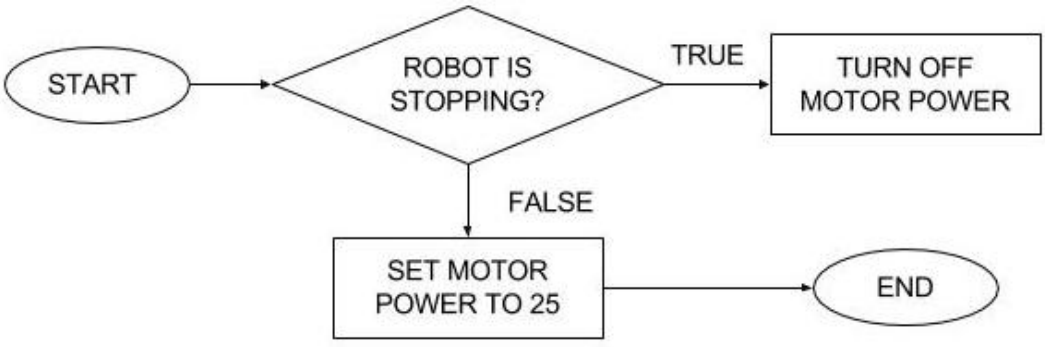



**Figure 10.3 Robot Drive function flow chart**

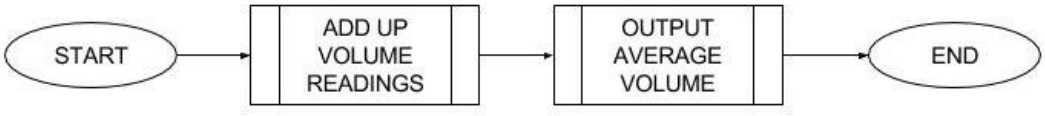



**Figure 10.4 Measure Volume function flow chart**

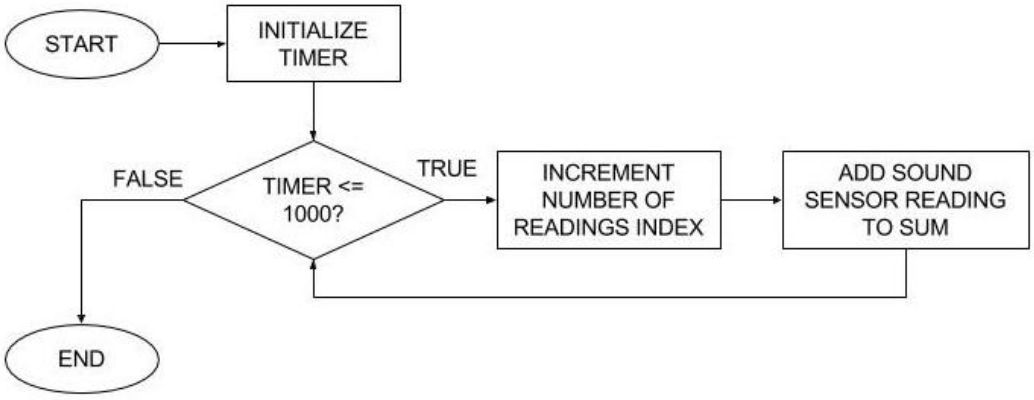



**Figure 10.5 Collect Readings function flow chart**

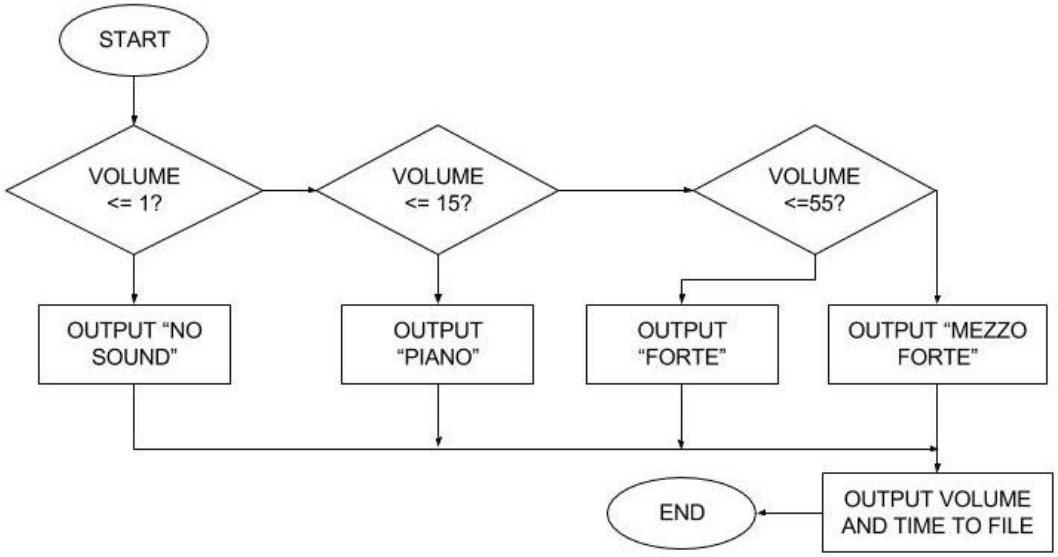



**Figure 10.6 Output Volume function flow chart**


**Figure 10.7 Pause function flow chart**