University of Waterloo

Faculty of Engineering

iScore

Design report of a Lego NXT Musical Score Scanner
Intended for Education

by
Chop'In Notes**:**
Justin Hsuanting Wang 20624376
Heerak Bae 20637172
Sung An Jung 20620139
Hsuan Ling Chen

1A Mechatronics Engineering

November 13, 2015

Table of Contents

List of Figures

Acknowledgements

This report is prepared by Chop'In Notes. The group would like to acknowledge Mechatronics TA, Jacob Shin for all the contributions that he has given since the conception of the project. Through his helpful suggestions and ideas, the completion of the project was made possible. The project lasted 3 weeks.

1.0 Introduction / Background

With increasing demands for musical education, a more accessible means of musical education is desired. iScore is a product that will address this need, and further information is detailed in this report.

1.1 Summary

This report details iScore, a multi-purpose music score scanner that targets beginning music students. The objective for this product is to create a robot that will scan and play the music score that the students will create. iScore serves the purpose to provide a more interactive method of learning music, and make learning music more accessible to different parts of the world. iScore implements three motors for its scanning operation. Each of these motors provide necessary movements that are needed for the scanner and the paper feeder. The chassis is designed with weight distribution in mind, and sensors were calibrated for precision. Some revisions were needed to earlier designs, due to limited material choice. It is concluded that iScore was able to meet all the constraints that were detailed in this report, and was also able to meet one of the criteria that enables users to create music score by hand. The implementations of the motors and sensors were effective, and iScore was successful in meeting the objectives. Use of more advanced color sensor, smarter control module and better materials is recommended.

1.2 Design Problem Definition

When learning music for the first time, many people experience difficulties if they can not properly reflect on their progress; without the aid of a teacher or professional, their ability to learn is hindered by their lack of knowledge and inability to distinguish correct and incorrect steps. This primarily stems from the principal's unfamiliarity with the concepts of "pitch" and "rhythm," and how it applies to the music being played. To properly hear what the music should should like, often times the only solution is to input the music onto expensive softwares which provide few personalized options and interaction.

1.3 Goals and Objectives

The goal is to provide a more interactive and exciting solution to beginners in music through iScore. iScore will read sheet music created by the user and play back the tune at a chosen tempo. By hearing what specific pitches and rhythms involved, the user can understand how to approach the music.

1.4 Constraints and Criterias

Initially in the design of iScore, several constraints were set to guide the design. Firstly, the robot must be able to playback music at a pitch range of at least two octaves, specifically pitches on the treble clef. Any less than two octaves may result in the playback of most pieces to be impossible. Secondly, the robot will be able to play from a tempo range of 60 - 240 beats per minute to allow for sufficient flexibility for the user. In addition, the robot needs to be able to play at four different length of notes. It must also read the score at a speed of at least ten notes/minute. The speed of the reader directly translate to its usefulness; the less time needed to wait, the quicker the user can benefit from the robot. Lastly, the robot needs to be able to distinguish all 12 key signatures and play the music accordingly, in order to avoid forcing the user to transpose the music directly.

Several criterias were also desired to help the robot's ability to aid in music. Firstly, to be able to play multiple notes simultaneously spanning multiple clefs, the robot can potentially playback chords on pianos and double stops on strings. The robot should be able to play all different rhythms (i.e. grace notes, appoggiaturas, triplets, and dotted rhythms) and be able to distinguish accidentals to allow for a broader spectrum of pieces. It should also read as it plays back sound, and be able to read from hand-drawn sheet music.

Throughout the design process, changes in the constraints were considered. The constraint to have two octaves of notes was often considered to be reduced, because the more notes that the robot must read, the more precise the motor encoder needs to be. However, through changes in how the motors functioned, the two octaves was considered to be sufficient.

Initially, it was considered possible to utilize all six of the colors from the color sensor. The white would be the default color, black as the end color, and the other four as different notes values. However, yellow proved virtually impossible to detect; therefore, the constraint has been changed to <u>three</u> different note lengths from four. A constraint that the robot must store 45 notes was added in to make it possible for the robot to play typical beginner tunes.

2.0 Hardware Design Implementation

The final design of iScore was decided through much testing and decisions. The hardware design has undergone significant revisions in certain design elements to ensure that the best and most efficient design is achieved with the limitation on available materials.

2.1 Motor Implementation

iScore uses three motors for its scanning operations. Each are used for the scanning operations, and also for feeding the paper to the robot. Motor 1 powers the scanning arm, which moves radially to scan each of the notes. A touch sensor is implemented on the side of the scanning arm, so that the motor encoder values will be reset every time the scanning arm completes its scanning cycle. Detail of this design can be seen in Figure 1 below.

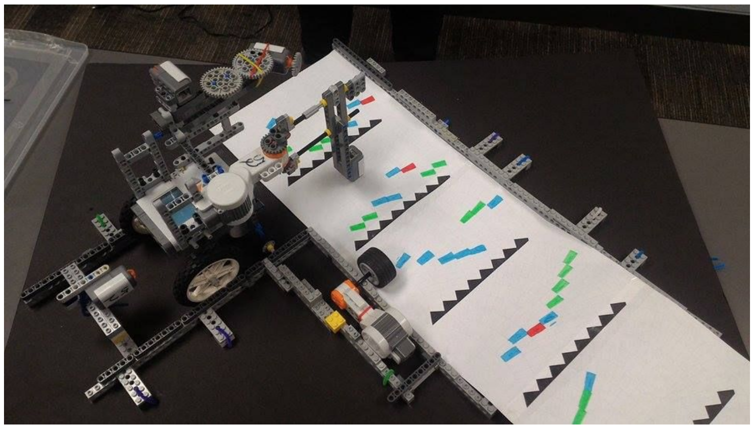
Figure 1: iScore Final Design

Initially, the scanning arm was designed to achieve vertical movement through the use of extending scanner arm and guiding rail that was built in front of the robot, as shown in Figure 2 below.

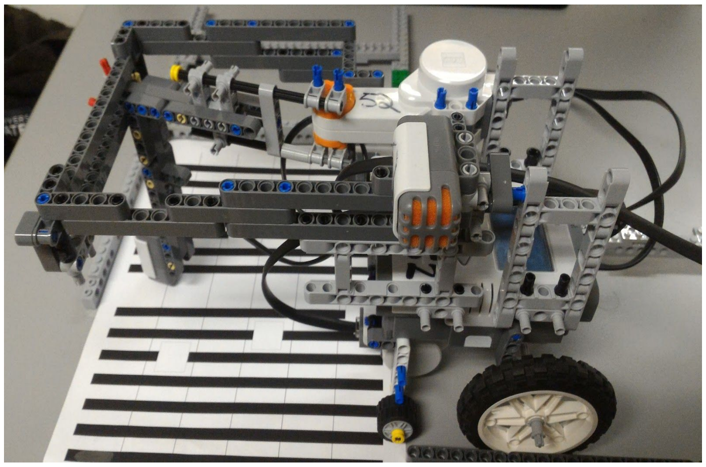

Figure 2: Earlier Design of iScore

However, this design was revisioned due to flexible nature of the lego material. Instead, the extending arm and the guiding rail was scrapped, in favour of direct radial movement of the scanner arm. Tradeoffs for this alternative solution is that the musical score is a little more difficult to create, and users would need some instructions.

Motor 2 is used to provide horizontal movement of the scanner, which moves the entire scanner assembly using wheels. This motor moves the scanner over to the next line of notes. The original rack design was adjusted due to the limitations on the quantity of racks provided in the lego kit. Alternate solutions given were to machine or 3D print these required parts. However, time constraint has restricted this procedure to be executed. Therefore, the wheel design was implemented. The compromise to this alternative design is that the scanner no longer moves perfectly straight due to the slight discrepancy in weight distribution.

Motor 3 is used to roll the paper to next batch of notes. This motor also uses wheels to push paper using friction. There were no design revisions made to this particular implementation.

2.2 Chassis Design

The chassis of iScore was designed with priorities on balanced weight distribution. The chassis of iScore has had one significant change to its original design. Since revisions were made in the method of scanning, the structure that enables extension of the scanning arm was removed, and the entire scanning arm has shifted towards the center for better balance. Since the wheel design is very susceptible to any imbalance, this design change brought significant improvements. The wheels lose traction if the chassis is front heavy, and any side weight difference can affect the direction.

2.3 Sensor optimizations

Since the color sensor is very susceptible to different lighting and shadow, the sensor location was decided through series of calibration and tests. The optimum distance between the color sensor and the colored surface was calibrated to be approximately 8mm.

The touch sensor was optimized by incorporating a bumper for a more reliable touch sensitivity.

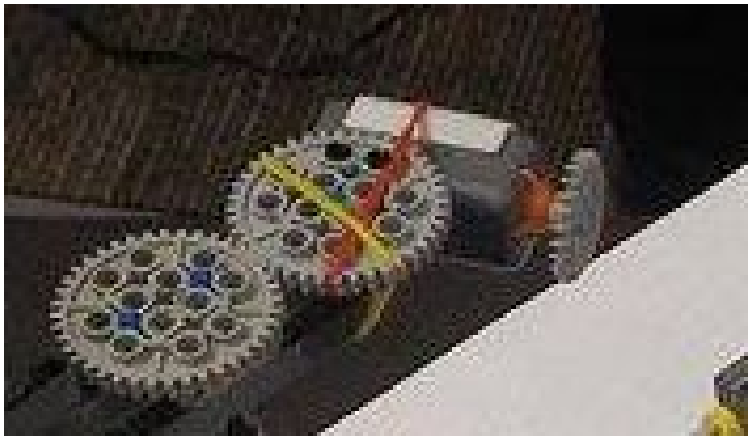

Figure 3: Touch Sensor

3.0 Software Design and Implementation

The final design of iScore was decided through much testing and decisions. The software design has undergone significant revisions in certain design elements to ensure that the best and most efficient design is achieved with the limitation on available materials.

3.1 Division and Testing of Software Design

While the software design depended on how the robot was mechanically designed, the way the code functioned and was divided up remained the same. The robot would store two parallel arrays of colors (integer values) and their respective motor encoder values through scanning a long sheet of paper, which would then be processed and outputted as music. The first major block is the mechanical aspect of the code, that is, the use of the motors and color sensor to scan of the notes (as shown in figure 3 below, left of double lines). This was further divided into parts for the two directions of movement for the scanner (motor 1 and motor 2), the rolling of paper (motor 3), and resetting the mechanical system. The second major component would be the processing of the raw values scanned to create an output (figure 3 below, right of double lines). This consisted of converting motor encoder values to frequencies, color to time, factoring in key and time signatures, using the sound sensor to set a tempo, and playing the notes.

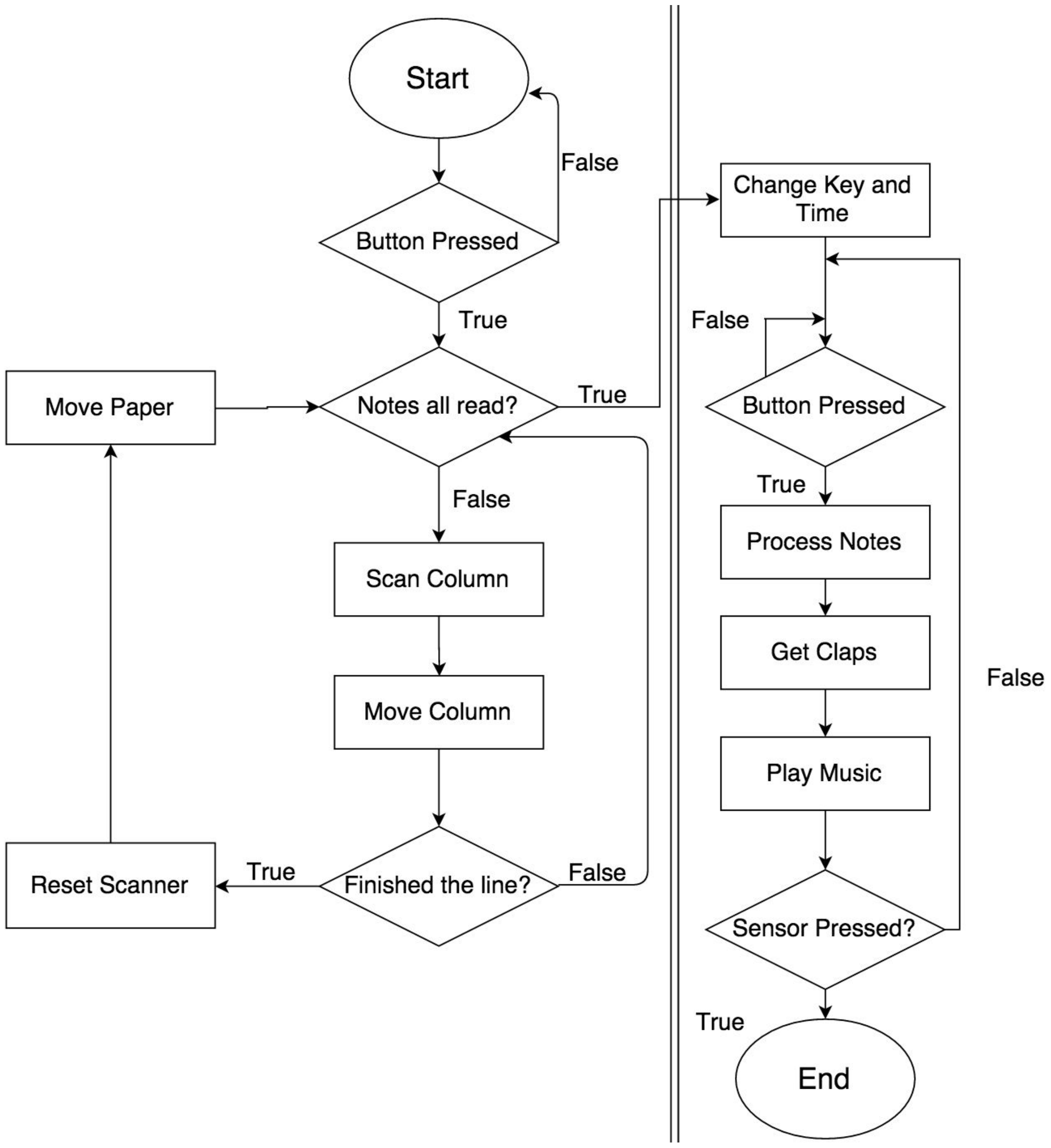

Figure 3: Software Design Flowchart

The reason for the split was for the testing and debugging process to be simplified. The mechanical portion could be tested independently of the processing portion, and vice versa. Testing the mechanical portion was done through changes in the track of the entire system, changing global variables (for encoder distances), and running one movement at a time. What values were actually stored is irrelevant until the mechanical system moved consistently. Similarly, by making test arrays of raw values, functions within the processing portion can be tested individually through analyzing the output and detecting and fixing discrepancies. Finally, the two parts are combined and thoroughly debugged by comparing what raw values are stored from the first half with what is needed to obtain proper output.

3.2 Problems, Decisions, and Tradeoffs

Multiple problems were faced during the software implementation. One of the biggest was accounting for the errors with the sensors and motors. Because of the way the sound sensor functions, which is through detecting changes in air pressure, the tolerance needed to detect claps, or sharp changes in air pressure, would vary from factors like ambient noise and proximity. The tolerance would have to be determined through repeated testing. Furthermore, the time intervals measured between claps would have large room for error; sound waves cause interference with one another, thus the changes in pressure reach the sensor and dissipate in inconsistent manner. With the motors and their respective encoders, every small error would propagate to huge inaccuracies, due to the repetition of functions. For instance, if the scanner would not travel in a straight line orthogonal to the rows of notes, the encoder values will begin to deviate. Similarly, if the paper roller did not roll the precise amount, the notes will no longer align with the robot. These discrepancies may be caused through incorrect weight distributions, variable paper traction (longer and heavier music score would be more drag), inconsistent encoders. To deal with these potential issues, repeated testing and the consideration of numerous variables was necessary, such as the length and orientation of the paper, the exact placement of the robot, weight adjustments, and incline of base. Once everything has been factored, a specific standard needs to be chosen, recorded, and followed for subsequent tests.

One notable decision made was how the frequency of the notes would be processed. In order to play music, the robot requires the exact frequency that it needs to output. Frequencies of notes is not easy to calculate as it runs through a logarithmic scale; each note has a frequency of a twelfth root of two higher than the last. In addition, the different modes of music (i.e. how the twelve chromatic notes of an octave are divided into 8 ascending notes like CDEFGABC) would mean that frequency between notes may change by a factor of a twelfth root or a sixth root. The approach of iScore is to have a default array containing two octaves or fifteen frequencies, following the key of a C major. This array will then be altered using a complex algorithm depending on the key signature that was inputted. For instance, if a key of G major (which has an F sharp) is inputted, all frequencies of F in the default array is increased by a factor of a twelfth root of two to raise the note to a sharp. This way when the notes are ultimately played, the key signature remains consistent and the notes are understandable.

An important tradeoff was necessary to satisfy the constraints of the design. In order to minimize potential errors, the music score could have included reference note at the beginning of every column. This would mean that the following notes could be calculated using relative motor encoder values to the reference, therefore less reliant on the consistency and accuracy of the motors and encoders for long pieces. However, this would mean that one in every eighth notes would have to be used as reference, therefore significantly reducing the number of played notes that could be stored. Given that the memory of the robot only allowed for parallel arrays of size 50, if every eighth were used as references, too few notes could be stored to play a typical piece.

4.0 Project Plan/Management:

The layout of the project planning is established with a Gantt Chart (See Appendix B) . The chart shows the list of tasks, their corresponding scheduled durations for it to be completed and their completion percentage. The project is split into smaller sections in order to distribute the tasks among group members so that better time management of the project can be met.

Finalizing hardware design took less time than planned in the design report. Therefore more priority and time was given other tasks such as software development and report write ups. However, due to the major change in the scanning arm hardware design, the software needed to be recoded to adapt to this change. Therefore, more time was spent on completing the software.

This section should include information on how tasks were split among the group members and list timelines for the project. Comment on differences between the project plan given in the design report and times and priorities changed during the project implementation

5.0 Conclusions

In conclusion, the intended objective for this design project is obtained. The robot meets all initially set constraints, since it is able to playback music at a pitch range of at least two octaves, and plays from a tempo range of 60 - 240 beats per minute. It also plays at least three different note values, and it reads the score at a speed of 12 notes per minute. iScore is able to distinguish all 12 key signatures and plays the music accordingly. Also, iScore meets one of the initially set criterias, since it is able to scan and play from hand-drawn music score. iScore has helped to provide a more interactive and exciting way for music beginners to understand the pitches and rhythms involved in a music piece.

6.0 Recommendations

One of the design improvements that can be made is to scan the notes vertically, and not radially. Although vertical scanning was a part of the initial design, it was switched to radial scanning due to limited material. Vertical scanning would improve the ease of use, and also increase compatibility.

Another improvement would be to incorporate racks instead of the wheel design. Rack design would significantly increase the accuracy of the scanning positions, and also increase the reliability of movement.

The use of more advanced and accurate color sensors could increase the number of different colors that can be detected. With the current robot system, only green, blue, and red can be detected reliably, therefore only three different note values are used. The more colors that can be used, the more variety in note values can be played, and the criteria of playing dotted and triplet rhythms could be satisfied. In addition, the colors can also be used as an indication of of accidentals, or a changes in pitch for that specific note only. This could mean that the robot could play chromatic music, satisfying another criteria.

By having access to more memory in the robot, the software design can implement reference notes, as described in 3.2. By having room to store the extra information provided from reference notes, the remaining notes can be much more accurate and still fulfill the criteria of properly playing lengthy tunes.

Lastly, the entire mechatronics system can include multiple NXT bricks, each scanning different sets of score. This can allow for the iScore to play multiple notes simultaneously, such as for piano music. In an even larger scale, the iScore would be able to play orchestral music.

7.0 References

[1] “What is a Gantt chart?,” *What is a Gantt Chart? Gantt Chart Information, history and Software*, 2012. [Online]. Available at: http://www.gantt.com/index.htm. [Accessed: Dec-2015].

APPENDIX

CODE:

```
const int widthScan = 200; // Total distance scanning
const int blockLength = 345; // Length the paper must be pushed (half a page)
const int MAXNOTES = 50; // Maximum number of notes

void scanColumn (int *height, int *color){ // Scanning using radial arm
   nMotorEncoder[motorA] = 0;
   motor[motorA] = -3;
   while (SensorValue[S3] == 6)
   {}
   *height = nMotorEncoder[motorA];
   *color = SensorValue[S3];
   motor[motorA] = 10;
   while (SensorValue[S4] == 0)
   {}
   motor[motorA] = 0;
}

void resetScanner (){ // Reset robot to initial state after scanning columns
   motor[motorB] = -10;
   while (SensorValue[S2] == 0)
   {}
   nMotorEncoder[motorB] = 0;
   motor[motorB] = 6;
   wait1Msec(400);
   motor[motorB] = -4;
   wait1Msec(200);
   motor[motorB] = 0;
}

void movePaper(){ // Moves paper by a half-page length for next set of columns
   nMotorEncoder[motorC] = 0;
   motor[motorC] = -30;
   while (nMotorEncoder[motorC] > -blockLength)
   {}
   motor[motorC] = 0;
}

void moveColumn(){ // Moves robot to scan next column
   if (nMotorEncoder[motorB] >= widthScan){
      resetScanner();
      movePaper();
   }
   else{
      motor[motorB] = 7;
      wait1Msec(560);
      motor[motorB] = 0;
   }
}

int heightToNote (int height){ // Converts motor encoder value into note value
   int starterPoint = -9;
   const int notewidth = 5;
   if (nMotorEncoder[motorB] < 30)
      starterPoint = -11;
   else if (nMotorEncoder[motorB] > 210)
      starterPoint = -14;
   else if (nMotorEncoder[motorB] > 170)
      starterPoint = -13;
   else if (nMotorEncoder[motorB] > 100)
      starterPoint = -11;
   else
      starterPoint = -9;
   height = height - starterPoint;
   displayString (1, "%d", starterPoint);
   return 1 + ( (-height) - (-height) % notewidth) / notewidth;
}

int readNotes (int *height, int *color){
   int count = 0;
   resetScanner();
   motor[motorA] = 5;
   while(SensorValue[S4] == 0){}
   nMotorEncoder[motorA] = 0;
   do{
   // Possible end conditions: no color, specific color, and specific height
reads
      scanColumn(&height[count], &color[count]);
      height[count] = heightToNote(height[count]);
      moveColumn();
      count ++;
   } while (color[count - 1] != 1);
       return count;
}

void buttonPressed (){
   while(nNxtButtonPressed == -1)
   {}
   while (nNxtButtonPressed != -1)
   {}
}
```

```
int keySignatureReader(int sigNote, int sharpFlat, int keyChanger) //TESTED
{
   int keyChange [7] = {0, 0, 0, 0, 0, 0, 0};
   sigNote = 11 - sigNote;
   int note = (int)round(1.7857 * (sigNote) - 1.5714) + sharpFlat;
   // Note: Approximation is accurate for our purposes
   if (note == 7) //Key of C major
   {}
   else if ((note != 0 && note != 10) && note % 2 == 0||(
      (note == 1 || note == 9) || note == 11)) //For sharp key signatures
   {
      while (note % 12 != 7)
      {
         if (note % 12 == 8 || note % 12 == 1)
         {
            keyChange[(int)round(0.5573 * ( (note % 12) - 1) + 0.8949) - 1] = 1;
            // Note: Approximation is accurate for our purposes
         }
         else
         {
            keyChange[(int)round(0.5573 * (note % 12) + 0.8949) - 1] = 1;
         }
         note = note + 5;
      }
      int placeholder;
      placeholder = keyChange[0];
      for (int count = 0; count < 6; count ++)
         keyChange[count] = keyChange[count + 1];
      keyChange[6] = placeholder;
   }
   else //For flat key signatures
   {
      while (note % 12 != 7)
      {
         keyChange[(int)round(0.5573 * ( (note + 6) % 12) + 0.8949) - 1] = -1;
         // Note: Approximation is accurate for our purposes
         note = note + 7;
      }
   }
   return keyChange[keyChanger];
}

void readKeySig (int *frequency, int sigNote, int sharpFlat){
// Converts notes based on the key signature
   int keyChange [7];
   for (int i = 0; i < 7; i++){
      keyChange[i] = keySignatureReader(sigNote, sharpFlat, i);
      displayString (i, "%d", keyChange[i]);
   }
   for (int  i = 0; i < 15; i++){
      frequency[i] *= pow(1.059463, keyChange[6 - ( (i + 3) % 7)] );
   }
}

int clapToBeats (int timeSig){ // Receives auditory input to calculate tempo
   int count = 0; // Sound sensor bounces at first
   int beats = 0; // Milliseconds per beats
   while (SensorValue[S1] < 60)
   {}
   wait1Msec(250);
   time1[T1] = 0;
   while (count < timeSig - 1){
      if (SensorValue[S1] >= 45)
      {
         count ++;
         beats += time1[T1];
         time1[T1] = 0;
         wait1Msec(250);
      }
   }
   return beats * 1.15 / count;
   //Multiplied by 1.15 to factor in error from sensor; bpm is 60000/return value
}

int color_time (int color, int beats){ // Converts colour read into note length
   if (color == 2) // Blue
      return beats;
   else if (color == 3) // Green
      return beats*0.5;
   else if (color == 4) // Yellow
      return beats*4;
   else if (color == 5) // Red
      return beats*2;
   return 0;
}

void playNotes
(int *frequency, int *height, int *color, int timeSig, int multi, int count){
   int beats = clapToBeats(timeSig);
   for (int  i = 2; i< count; i++){
      int time = color_time(color[i],beats)*multi/5;
      playImmediateTone (frequency[height[i]], time);//heightToNote(height[i],
i%8)
      wait1Msec(time);
      clearSounds();
      wait1Msec(time/4);
   }
}

task main (){
   SensorType[S1] = sensorSoundDB;
   SensorType[S2] = sensorTouch;
   SensorType[S3] = sensorColorNxtFULL;
   SensorType[S4] = sensorTouch;
   wait1Msec(2000);
   int height[MAXNOTES], color [MAXNOTES];
   int frequency [15] = {989, 880, 784, 698, 659, 587, 523, 494, 440,
      392, 349, 330, 294, 262, 247}; // The frequency of notes from B3 to B5 in
Hz
   buttonPressed();
   int count = readNotes(height, color);
   readKeySig (frequency,height[0] - 1, 0);
      // Blue = flat, green = natural, yellow = sharp, a height of F (10) is F
major)
   int timeSigBar = height[1]; // Height is number of beats
   int timeSigBeat = color[1] * 2; // Blue 4, Yellow 8
   buttonPressed();
   do{
      while(nNxtButtonPressed!=-1){}
      playNotes(frequency, height, color, timeSigBar, timeSigBeat, count);
   } while (SensorValue[S2]==0);
}
```