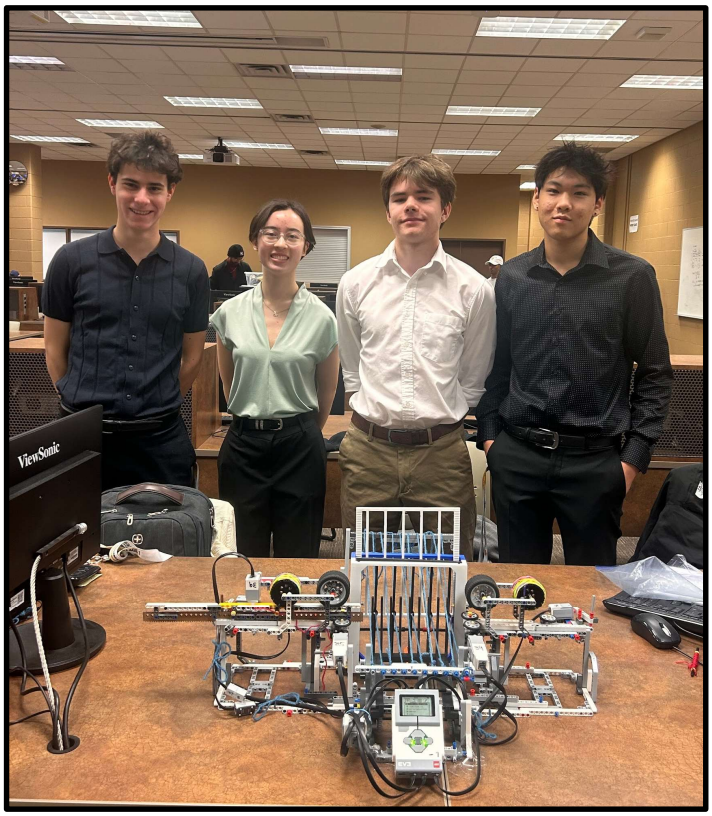

Automated Loom

Robot 8-14

Liam Dachner 21056460

Ethan Guan 21066982

Alanna Rudolph 21063864

Josh Stadnyk 21054733

MTE 100 & MTE 121

Dec. 5, 2023

Foreматter

Summary

This project aims to develop an automated loom capable of independently weaving a variable length of fabric out of yarn using a Lego Mindstorms EV3 kit. The team was able to successfully create a robot that fulfilled all constraints and criteria.

The automated loom was made up of four major mechanical parts: the fork, heddles, shuttle, and yarn control. A major challenge was the complexity of a free floating moving part that had to span a gap with no supports. Additionally, the team modelled and 3D printed some custom parts for this build.

Acknowledgements

The team would like to thank the MTE 100 TAs for all their help and support. Without the feedback and advice they received, it would have been very difficult to complete this project.

Table of Contents

List of Figures

List of Tables

1.0 Introduction

Although the EV3 Lego Mindstorm set is often used to make robots that move around a space, this team decided to take a different approach. The goal in making a loom was to create a robot that would instead create a product, such as a woven bracelet, which could be made live on demo day.

1.1 Description of Problem

The loom involved both mechanical and computational complexity. On the mechanical end, so many moving parts created countless problems and iterations of design. Computationally speaking, the numerous failsafe and emergency systems that were built into the code served as both a necessity when working with so many loose threads and a challenge to write.

Through a highly iterative design process and much trial and error, the loom was eventually completed and fully successful. This report documents both the struggles and successes of this process and the product that was eventually created.

1.2 Background Information

There are several key pieces of loom vocabulary that are key to understanding the inner workings of a loom and this report.

The warp describes the threads or yarns that are stretched across the loom and the weft is the thread or yarn that is passed back and forth through the warp threads [1, pp.14]. The warp and the weft are woven together by the shuttle, which holds the weft threads and is used to move the weft through the warp threads [2]. The heddles, which hold the warp threads at one end, move them up and down after the shuttle has passed through [2]. The shed is the vertical space between the two sets of warp threads that the shuttles pass through [1, pp.14].

2.0 Scope

The robot created a fabric with a user-specified number of rows. To do this, the robot first took input via the buttons on the EV3 Brick. Users could select the number of rows they wanted to weave by using the up and down buttons.

The robot then received input from two touch sensors, a colour sensor, and the motor encoders. The touch sensors were used to detect when the shuttle had completed a row and count the number of completed passes of the shuttle that carries the thread.

The colour sensor was used to detect whether or not there was still string on the spool. If the sensor could detect red, the colour of the thread, the program continued to run. If the spool ran out of thread, or the sensor could no longer detect red, the program paused.

The motor encoders were used to set the distance that the heddle and the fork move. The heddle and fork moved to and from this distance over the duration of the program.

Motors were used to power several features of the robot. The shuttle was pushed across from one side by a motor using a rack and pinion system. Once it reached the other side of the gap, the shuttle connected with a gear powered by a second motor to complete the pass.

The heddle was controlled by one motor attached to a gear. One piece of the heddle was lowered while the other was simultaneously raised. The fork was pushed by a motor until it hit the predetermined distance set by the motor encoder.

The robot tracked how many rows of thread had been woven. Once this reached the number of rows the user inputted, the robot began the shutdown procedure. During the shutdown, all motors turned off, a message informing the user the task was completed was displayed, the elapsed time was displayed, and the robot waited for the enter button to be pressed. Once pressed, the program ended.

There were several changes in scope from the beginning of this project. The first choice the team had to make was the maximum number of warp threads. This determined the width of the loom and the fabric

that could be produced. The team chose to use ten warp threads, since it was a good balance between small enough to reduce complexity and large enough to demonstrate proof of concept.

Another change of scope was choosing to use only four motors. The robot could have used at least five motors, but to avoid using a multiplexer, it was built with four. This meant the heddle had to be designed to operate using only one gear as a power source.

The final change of scope was deciding to use only red yarn for the weft. It is not entirely realistic to have a robot that can only weave red fabric; however, discussions with Ryan Consell revealed that red is the colour that the colour sensor can detect best. Therefore, to avoid potential hardware errors, exclusively red thread was used.

3.0 Constraints and Criteria

3.1 Constraints

Although the robot had many moving parts, the team decided to only use four motors as a constraint. It was possible to request the motor multiplexor, but the team thought it would add extra complexity to the robot and its code. Because of this, the mechanisms had to be designed to work with as few motors as possible. This constraint did not change over the course of the project. It was helpful to the project because it forced the team to maximize the use of resources and make more efficient designs.

A constraint the team had to work around was a free-floating gap that the shuttle had to pass over, highlighted in red in Figure 1. This area had to remain empty because otherwise, it would interfere with the warp threads when the heddles flipped. It meant the robot had to pass the shuttle with no supports above or below in the middle and catch the shuttle after it passed over the gap. This constraint did not change throughout the project.

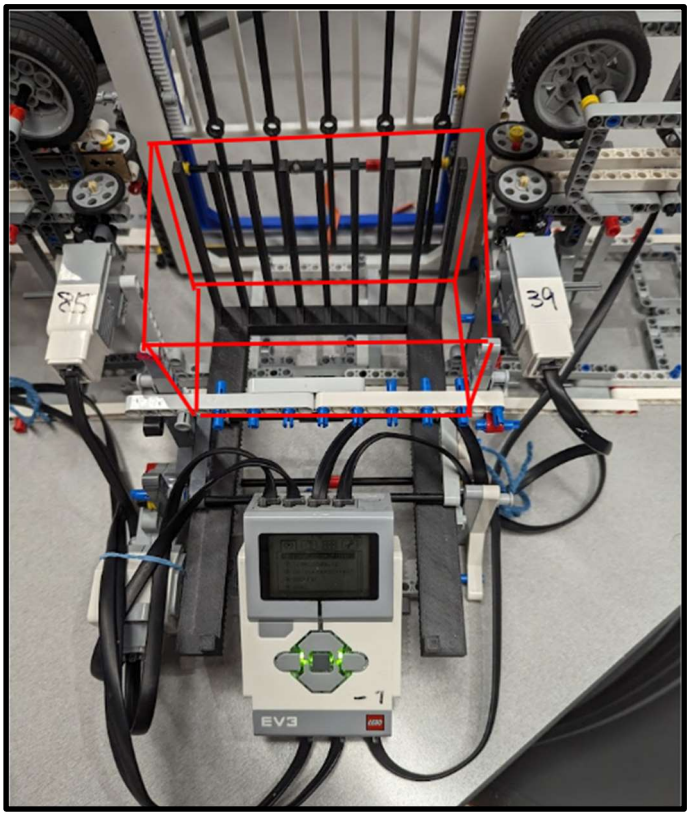

Figure 1: Free space the shuttle must pass through.

3.2 Criteria

During our initial design discussions, the team decided that they wanted the robot to weave with at least 10 warp threads. More warp threads directly increased the complexity of our robot because it increases the gap discussed in 3.3. The team agreed that ten warp threads would allow the robot to produce a sizable piece of fabric, while also being feasible considering the timeline and resource limitation. This criterion was not changed throughout the project.

The team also wanted the user to be able to choose how many rows of weft thread the user wanted to weave. This was important to the project since any real weaving robot would have some degree of customizability. A loom that could only weave a set number of rows would be very impractical. This criterion helped guide much of the software because it meant that most of the code was dependent on user input and most of it couldn't be hard-coded. The criteria changed slightly throughout the project. In the final robot, the user was only able to choose rows in multiples of two, because one cycle of the robot wove two rows of thread.

4.0 Mechanical Design and Implementation

4.1 Fork Mechanism

The fork is a sub-system of our robot. Its responsibility is to compress the weft threads after each pass of the shuttle. Designs of the fork stayed mostly constant throughout our project. The idea was to have a series of prongs that extended upwards, going in between the warp threads, and a mechanism to extend and retract the prongs. Initially, the team thought of attaching the fork to a motor as shown in Figure 2.

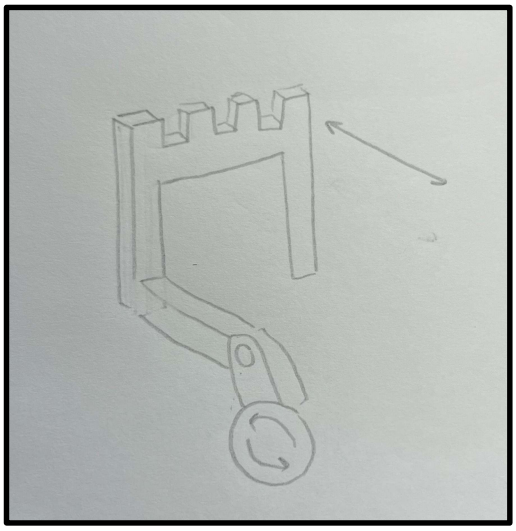

Figure 2: Sketch of fork idea

The second iteration of the fork design instead used a wheel that turned and used friction to move the fork piece, as shown in Figure 3.

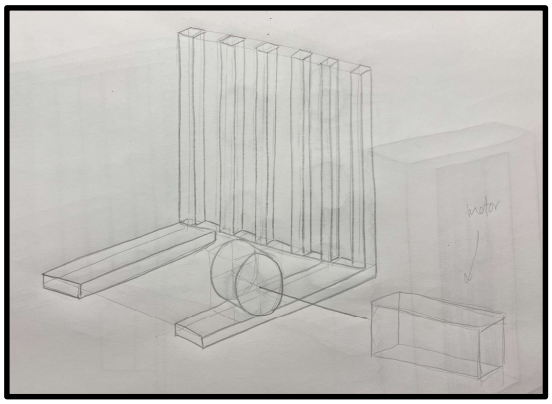

Figure 3: Another sketch of fork idea

The first design used Lego axles as prongs and wheels to extend and retract the fork, shown in Figure 4 and Figure 5. This design verified the concept of the fork but had many problems. First, the fork didn't

extend nearly as far as we needed it to. The wheels used to extend and retract the fork were also inconsistent since the fork could slip against the wheels, causing it to be misaligned. The prongs were different sizes since we had a limited supply of axles, and the gap between the prongs wasn't very adjustable since they had to line up with the holes in the Lego.

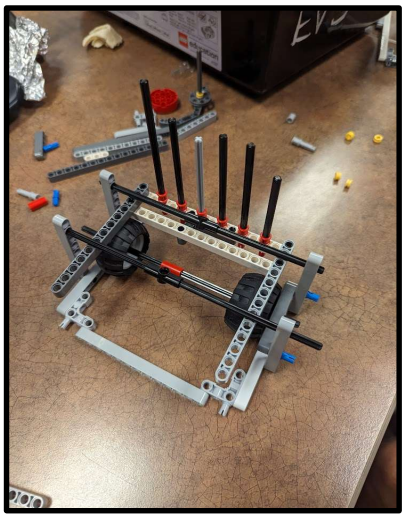

Figure 4: Prototype of fork

Figure 5: Prototype of fork

To solve these problems, the team decided to 3D model and 3D print the fork, shown in Figure 6. This gave more control over the size, length and spacing of the fork. Additionally, it was decided that a rack and pinion mechanism would be more consistent than wheels. Wheels were switched out for spur gears and the fork was modelled to have a rack on its underside. The 3D printed fork was also sturdier than the Lego components as there were no points of connection, which is a weakness for Lego pieces. The downsides of 3D printing were that it was time consuming and cost some money. 3D prints are also not

easily modifiable, which proved to be a problem later on. To mitigate these downsides, the team printed the part early on and joyously discovered that the cost of the full print was less than a dollar.

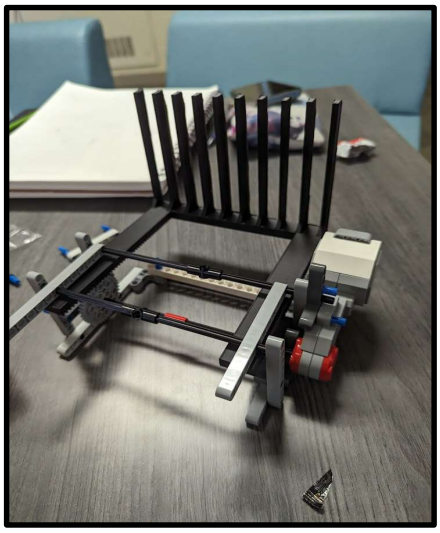

Figure 6: First iteration of 3D printed fork.

After some testing, there were still some issues with the fork. There was one too many prongs due to a miscalculation. Because the prongs go in between the warp threads, only nine prongs are needed for ten rows of thread. The fork still didn't reach far enough, and there wasn't a part to stop the fork from overextending and falling off the gears once it had reached its max length. Since 3D prints aren't easily modifiable, the team had to 3D model and print a new fork. The final version of the fork, shown in Figure 7, had nine prongs, was 20 cm in length, and had backstops to prevent it from over-extending. The rest of the fork mechanism remained the same.

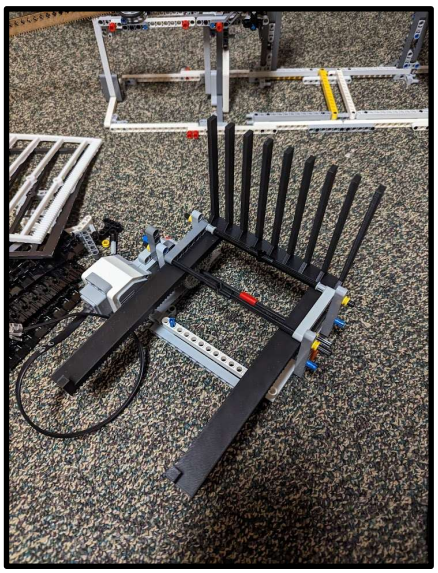

Figure 7: Second iteration of 3D printed fork.

This design worked well and was consistent, so it was kept for the final design. It was connected to the main frame with 2 L-shaped connectors (Figure 8).

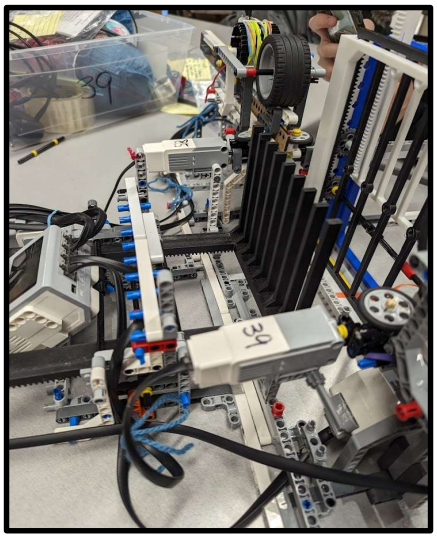

Figure 8: Fork side view

4.2 Heddle Mechanism

The function of the heddle was to swap the warp threads from the upper position to the lower position and vice-versa after each pass. Potential options to move alternating warp threads included a heddle bar, a pulley system, and a double rack and pinion system. A heddle bar posed challenges in that it would be more difficult to ensure there would be enough clearance for the shuttle to pass through. The team decided a pulley system might be too complex to build with only one motor, as described in 3.1 Constraints. The chosen design used a double rack and pinion. This allowed opposite threads to move up and down with only one motor. A sketch of one heddle is shown in Figure 9.

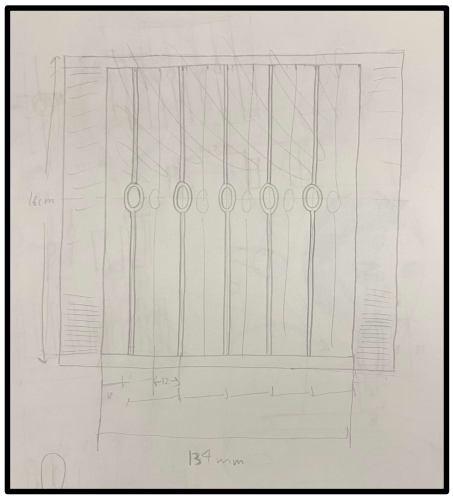

Figure 9: Initial sketch of heddle piece

4.2.1 Heddle Plate

After initially sketching different drafts for the heddle, the team 3D modelled the heddle in SolidWorks, shown in Figure 10. The design included linear gears on the ends and five beams for each warp thread. Additionally, the first beam was specifically spaced from the side so two identical copies could be printed at the same time. One of these copies would be flipped around to create an offset pair of heddles.

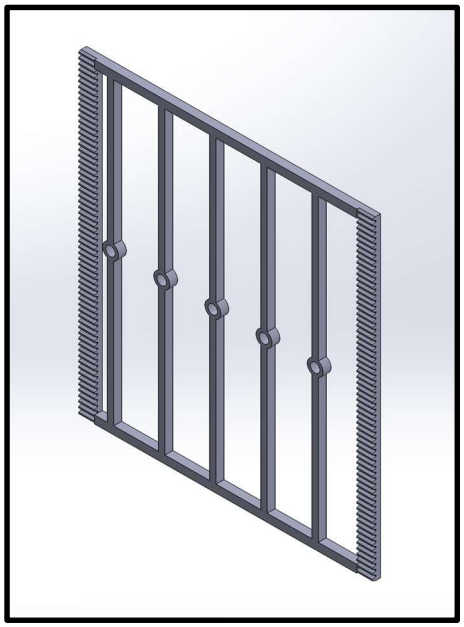

Figure 10: 3D model of heddle piece

With both heddle plates printed, a few gears were attached to the skeleton of the loom so that the team could observe how the heddles would function. The team immediately ran into a problem; the pieces didn't stay together, and many hands were needed to hold them.

4.2.2 Casing

To solve this issue, the team decided to design a frame (Figure 11) to securely hold each heddle and prevent it from moving in any direction other than up and down. The team modelled the case in such a way that it could easily be 3D printed without the need for supports. Each half of the case was connected with friction-fit pegs. In addition, through holes were needed along the side of the case to connect the heddle system to the rest of the Lego robot. As seen in Figure 12, the team originally designed the frame with a rail system. However, it was found to be unnecessary as the heddle plates fit well enough that they did not need more support and the rail created too much friction for the heddle to slide freely. The team chose to reprint the heddles without the rail.

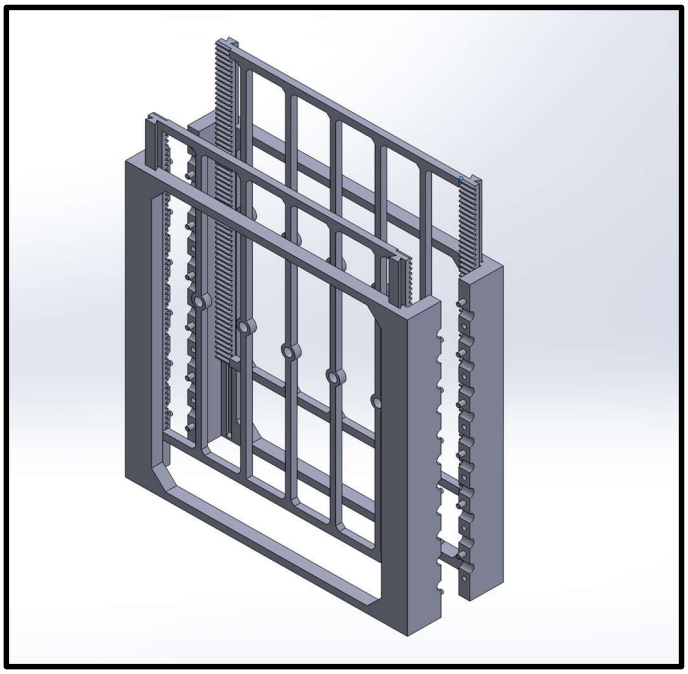

Figure 11: 3D model of heddle with case

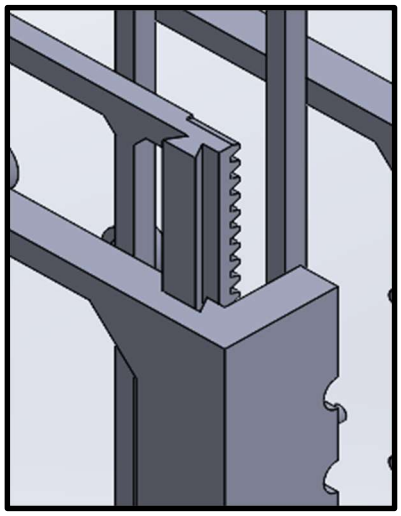

Figure 12: Close up of rail on heddle case

4.2.3 Heddle Implementation

Four 16-tooth gears were used between the heddle plates. Two of the gears were connected onto a single driveshaft that was powered by one motor. As shown in Figure 13, the heddle had to be connected to the loom so that the fork and warp threads did not interfere with each other. This was mainly done through the use of spacers to accurately position each component.

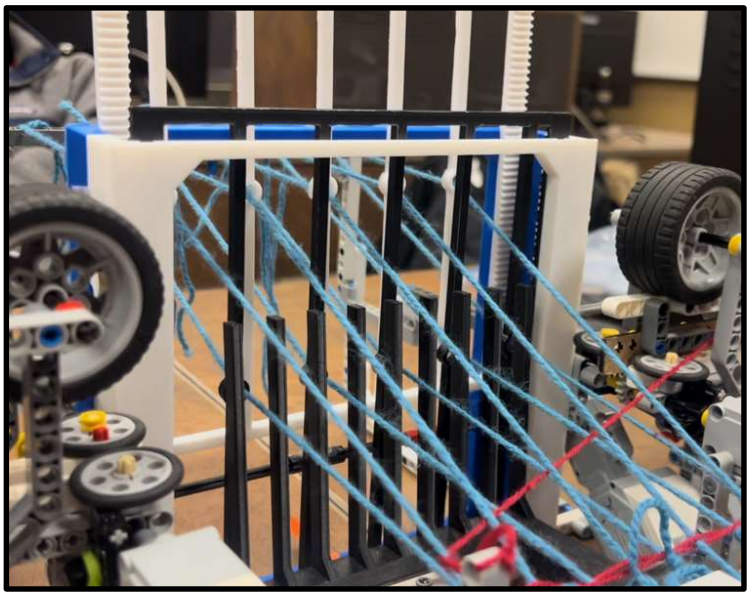

Figure 13: View of yarns in the heddles

4.3 Shuttle Mechanism

The shuttle passed the weft yarn through the shed to weave the fabric. The shuttle must span a large gap so that it does not interfere with the upwards and downwards motion of warp threads.

4.3.1 Track

The track was one of the trickier components to build, as the shuttle had a tendency to fall either to the left or to the right. To correct this tendency, small bumper wheels were added on the right and left side of the first contact point on either side of the gap (Figure 14).

Large bumper wheels were added to the top of the supports on either side (Figure 15). These large bumper wheels created a downward force on the shuttle and held it tightly in an upright position. The second wheel was wrapped in elastics to increase the radius of the tires and therefore provide additional downward force.

Despite these two sets of supporting bumper wheels, the shuttle continued to fall over. As a final adjustment, two long horizontal Lego beams were added on either side of the track. (Figure 16).

Figure 14: Small bumper wheels

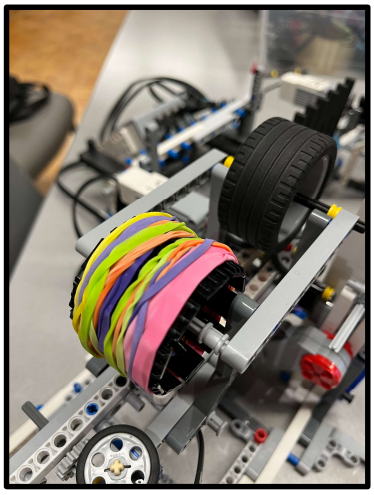

Figure 15: Large bumper wheels

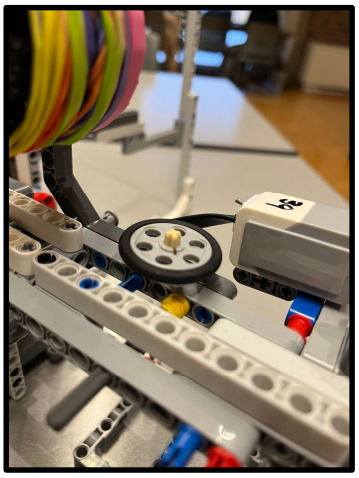

Figure 16: Track

4.3.2 Touch Sensor

The team used two touch sensors, one at each side of the track, to detect when the shuttle reached the end of a row. The initial plan was to place each touch sensor at the far end of the track and have the shuttle directly press it. The team found this would have made the overall structure very long, so instead a lever mechanism was used (Figure 17). It interacted with the spool, which was midway on the shuttle, which meant the team did not have to build the structure longer in order to support the touch sensor.

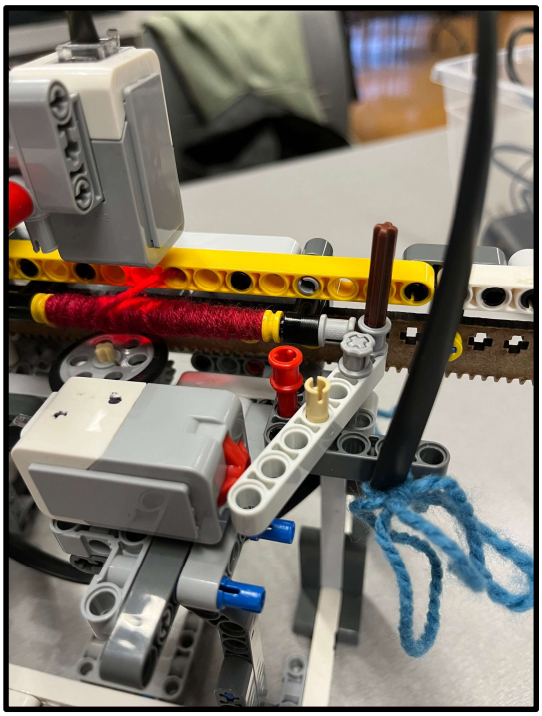

Figure 17: Touch sensor with lever

4.3.3 Colour Sensor

The team selected red as the weft yarn colour, as it was most easily detected by the colour sensor. The colour sensor was mounted on one side of the supports over the end point position of the spool (Figure 18). When the spool was out of thread, the colour sensor would no longer detect red and request that the spool be refilled. The only difficulty was determining what distance between the thread and the colour sensor was optimal. This process is outlined in 5.4.1 Unit Testing.

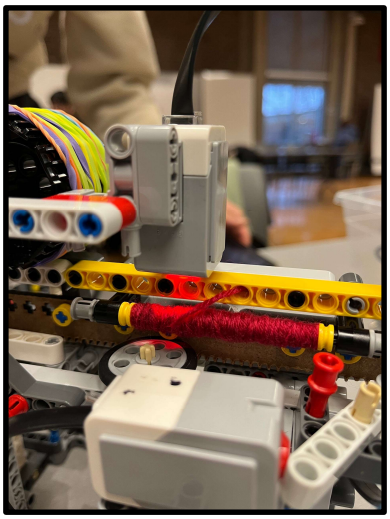

Figure 18: Colour sensor positioned over spool.

4.3.4 Spool

One challenge the team faced when trying to make the shuttle run was that the yarn initially didn't unravel smoothly. At the furthest ends of the track, the yarn was being pulled almost parallel to the spool, which meant it didn't unravel well and created a lot of tension on the warp threads.

To solve this, the team drew inspiration from sewing machines. When threading a sewing machine, the thread goes through multiple changes in direction in order to maintain good tension. The team implemented this idea by feeding the yarn through a hole in the Lego beam on the shuttle (Figure 19). This helped the tension issue by limiting the angle at which the yarn could be pulled off the spool.

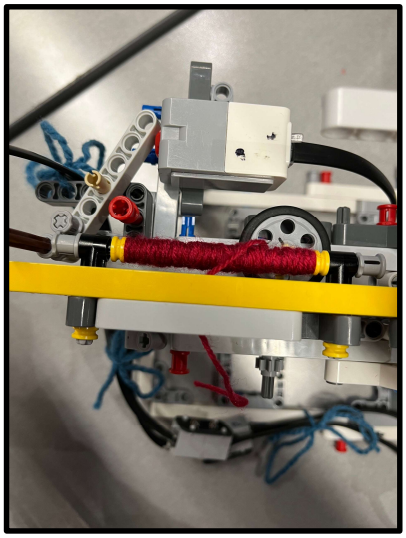

Figure 19: Yarn from spool threaded through beam.

4.4 String Management

The team found that if there wasn't sufficient tension on the warp threads when the heddles were in each position, then the weft thread would pull the warp out of position.

The warp yarns were attached to the loom at two points: over the fork and behind the heddles, as shown in Figure 20 and Figure 21. Initially, the back yarn holder was placed an arbitrary distance away from the heddles. The team discovered that this caused different tension in the strings depending on which position the heddles were in. Also, the position of the front yarn holder could not be changed since it had been placed to optimise the size of the shed for the shuttle to pass through.

The team decided the best place to put the back yarn holder was so that the distance between the heddles and back holder was the same as the distance between the heddles and front holder. Additionally, it was built at a height such that the warp threads formed a parallelogram shape when viewed from the side. This ensured the tension in the warp threads was the same at the top and bottom points of the heddles.

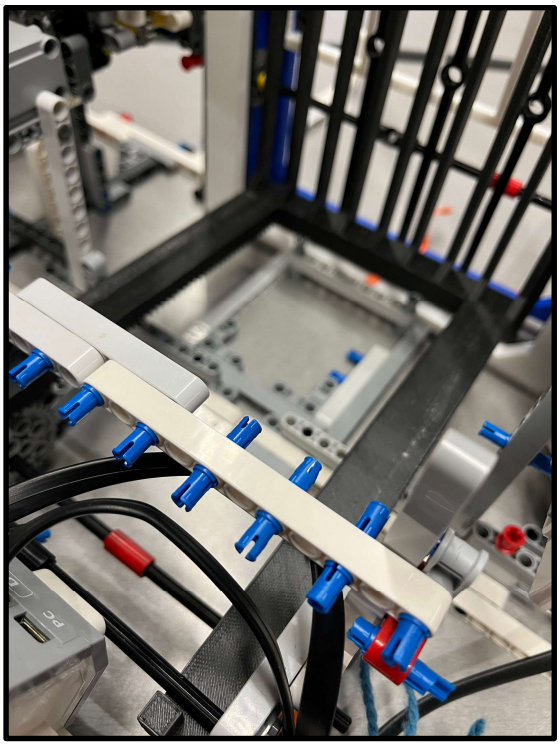

Figure 20: Yarn holder attached to fork.

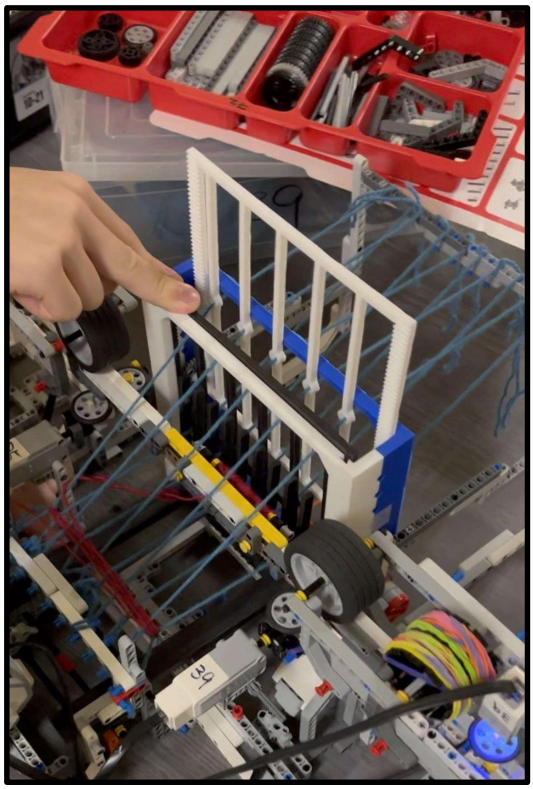

Figure 21: Back and front yarn holders

4.5 Overall Assembly

All the moving parts had to coordinate together. The main task was to ensure the warp threads, heddles, and fork aligned properly. Ideally, the warp threads would be perfectly perpendicular to the weft without bending where they were held by the heddles and the fork tines would sit right in the middle between each warp thread. The team achieved this to a satisfactory level mostly by trial and error through adjusting the fork and yarn holder lateral positions.

As the structure was quite large, the team ran into some issues managing cables. Longer cables were used for the sensors furthest from the brick, such as the touch and colour sensors, but there still was a lot of tension in the cables and it made the structure look messy. The team chose to used cable extenders instead, and used rubber bands and string to neatly organise long cables, as shown in Figure 22 and Figure 23.

The last issue faced by a larger structure was that the team frequently ran out of Lego parts such as connectors. Despite requesting additional parts multiple times, it was necessary to optimise the build to use pins and beams only where they were most needed.

Figure 22: Elastic band organising cables.

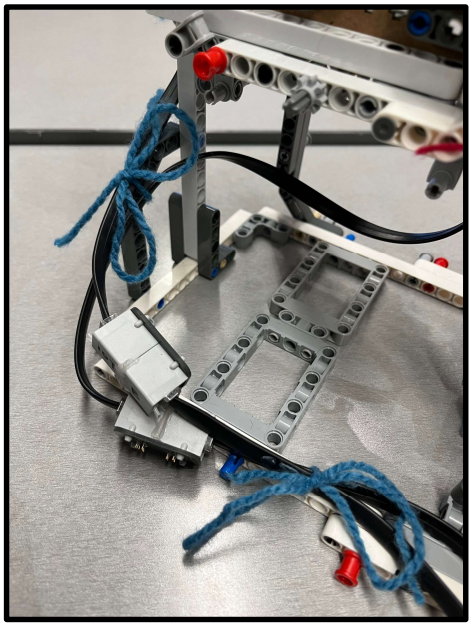

Figure 23: Yarn ties organising cables

5.0 Software Design and Implementation

5.1 Flow Chart

The software was divided into four blocks: start up, shut down, normal operation, and exceptional operation. The team chose to break up the program this way since the main work was writing normal operation and start up and shut down were specific cases of exceptional operation. Figure 24 shows a high level overview of the start up, shut down, and normal operation.

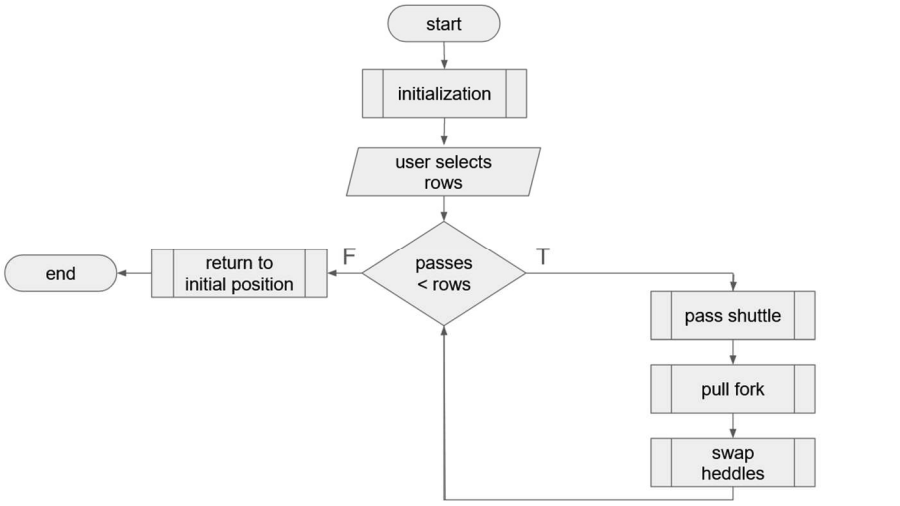

Figure 24: High level flow chart of normal operation

5.2 Task list

The team used the following task list for our demo:

Start up:

- Ask user to input # of rows
- Store number of rows

Normal Operation/List of tasks:

- Pass shuttle until colour sensor is hit
- Pull and return fork
- Flip heddles
- Return shuttle until colour sensor is activated

- Repeat for n number of rows

Operates/reaction to environment:

- Colour sensor senses when we are out of string
- Touch sensor counts the number of passes
- Motor encoder moves the heddle and fork to the desired distances
- Emergency pause feature can be used as user wants

Shutdown:

- Program will end once all rows are woven and the Enter Button is pressed.
- The display will output the time it took to complete the task (in seconds)

The original task list did not include some "extra" features that were not required for a minimally viable product, such as the emergency pause feature or the colour sensor detecting when the spool was empty.

5.3 Functions

Table 1 describes each function, the author, and its use.

Table 1: Overview of functions

Function name	Author	Return type	Parameters	Description
configureAllSensors		void	-	Configures all sensors.
waitButton		void	TEV3Buttons button_name	Trivial function to detect whether a button was pressed.
getRows	Alanna	int (number of rows)	-	Prompts user to enter the desired number of rows using buttons and returns the chosen number of rows. Calls waitButton.
flipHeddles	Josh	void	bool cw	Swaps the two heddles.

pushFork	Ethan	void	int pass	Pushes the fork back a distance depending on the number of rows already woven to compress string.
passShuttle	Liam	void	int pass	Moves the shuttle across until it hits the touch sensor. Calls flipHeddles, pushFork, and checkEmergencyPause.
checkEmergencyPause	Josh	void	-	Checks if right and left buttons are pressed at the same time. When buttons are pressed, all motors are stopped and current motor encoder values are stored in an array.
checkSpool	Ethan	void	-	Detects whether spool has string or not. Display tells the user to refill the spool.

5.4 Testing

5.4.1 Unit Testing

Each of the robot's nontrivial functions were tested individually using unit tests, outlined in Table 2.

Table 2: Function testing methods

	How function was tested	Criteria to pass test
getRows	- Displayed the number of rows received while testing in the EV3 simulator before integrating. - Function was run several times with different row	- Pressing the up button results in the displayed number increasing by the correct increment - Pressing the down button results in displayed number decreasing by correct increment

	selections. - The user tried to go below the set minimum and above the set maximum.	- If pressing buttons resulted in the number of rows going out of bounds, then row number didn't change - After user confirms number of rows, the correct number is displayed back to the screen.
flipHeddles	- Function was run 10 times.	- Raised heddle touches bottom after flip - Lowered heddle is fully raised after flip - Motor encoder successfully triggers subsequent heddle flip.
pushFork	- Function was run 30 times	- Fork fully retracts - Fork fully extends - Motor encoder successfully triggers subsequent fork push - Fork retracts 0.2mm less time
passShuttle	- Function was run 20 times consecutively	- Shuttle moves to the right without derailing or getting caught - Shuttle successfully activated right touch sensor and stops. - Shuttle moves to the left without derailing or getting caught - Shuttle successfully activates left touch sensor and stops.
checkEmergencyPause	- A test program was run with a motor spinning at various speeds. - User attempted to pause the motor 10 times	- Motor stops spinning when the left and right buttons are pressed - Motor resumes spinning at the previous speed when the left and right buttons are pressed again.
checkSpool	- The function was run, and red yarn was put periodically under the colour sensor - The enter button was pressed with and without red thread under the colour sensor.	- Nothing happens if red thread is detected - Screen displays "Please refill thread" if no red is detected. - If the enter button is pressed with no red thread under colour sensor, nothing happens - If the enter button is pressed with red thread under colour sensor, the screen is cleared.

5.4.2 Integrative Testing

To ensure all the functions worked with each other, the robot ran for 50 rows of thread. The team counted the number of rows woven after completion to ensure it was the correct amount. Each function had to

execute after the previous one in the correct order, meaning the trigger events such as touch sensors or colour sensors were leading to the right functions. The checkEmergencyPause function was tested by pressing the left and right buttons during each stage of normal operation, such as the shuttle, heddles, or fork moving.

5.5 Data Storage

The program used variables to store the number of passes, booleans to store the direction of the heddles and shuttle, constants for gear diameters and distance to radian conversions, and arrays to store motor values for the pause function.

6.0 Verification

The team met the constraint of having only four motor slots as the robot achieved everything in scope while only using two small motors and two large motors. The team also met the constraint of leaving open space for the warp threads and the shuttle successfully passed over the gap with supports only on either side.

The team met the criteria of having at least ten warp threads and was able to demonstrate the robot weaving with them. The team also met the criteria of allowing the user to input the number of rows of weft thread for the robot to weave, because the complete loom used button input from users and successfully wove the correct number of rows.

7.0 Project Plan

7.1 Timeline

The team had a timeline for the robot outlined in Figure 25. There were no revisions to the timeline or the project plan. The timeline was followed very closely, but there were times when certain tasks took longer than expected and the team was a day or two behind the timeline. An example of this was with the full tests, which took an extra day since some unexpected errors occurred when everything was put together.

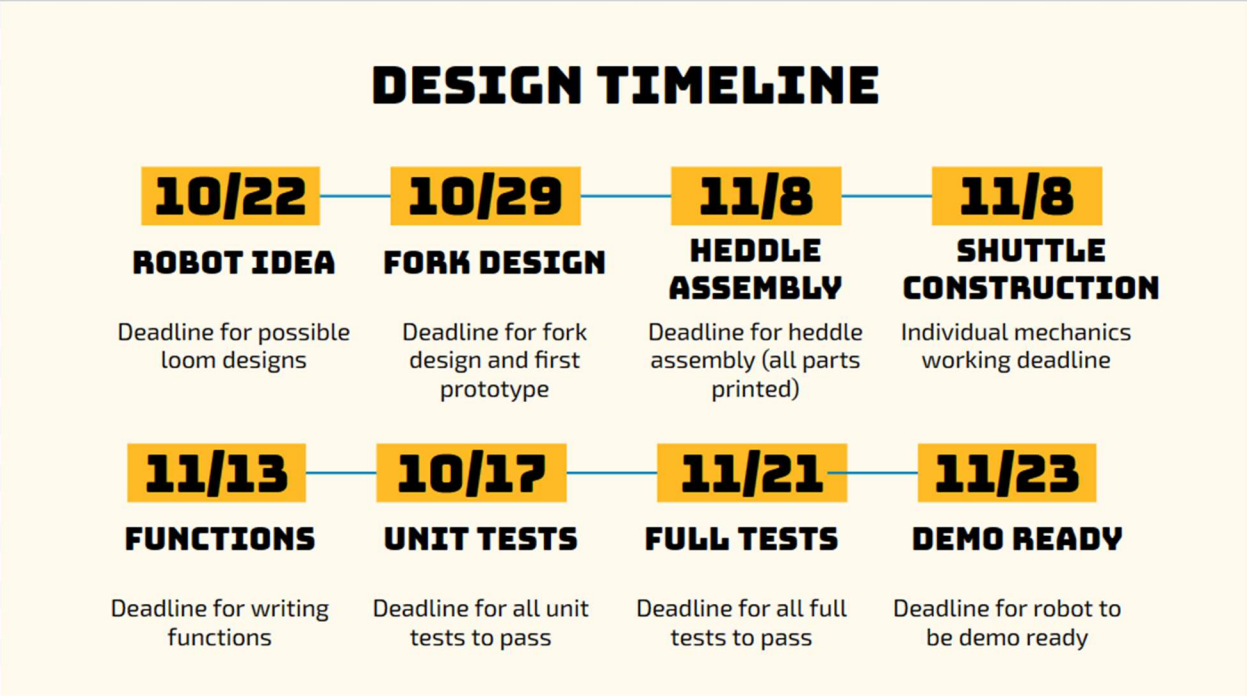

Figure 25: Design timeline

7.2 Division of Tasks

Table 3 and Table 4 show the division of tasks between group members.

Table 3: Mechanical tasks

Subtasks	Description	Task Dependencies
Fork Mechanism (Ethan)	- compresses the weft threads whenever the shuttle is passed back	- the spacing between each prong had to be equal to the spacing of the

	and forth through the warp threads. - after every compression, the fork needs to return 2mm less to account for the added string.	heddle beams
Heddle Mechanism (Josh)	- flips the warp threads from up to down and down to up - uses a rack and pinion system while only using one motor.	- had to be spaced properly with the fork so there is no interference - the gap between the upper and lower threads had to be large enough for the shuttle to pass without interruption
Shuttle Mechanism (Liam)	- a “free-floating” beam that carries the weft thread back and forth. There cannot be anything supporting the shuttle when it passes through the warp threads. - uses touch sensors to detect when it is at the end of a pass - uses a colour sensor to detect when the spool is out of string.	- had to be one of the first parts completed as the team needed to know the length of the gap the shuttle would have to pass
String Management (Alanna)	- the warp threads needed to be evenly spaced and they had to have equal tension in both the upper and lower positions. - the spool on the shuttle had to easily release the weft thread so the shuttle would not get caught while weaving.	- the last part of the loom to be completed.

Table 4: Software tasks

Subtasks	**Description**	**Task Dependencies**
User Input - Interface (Alanna)	- asks the user how many rows they would like to weave. - uses an upper and lower bound for how many rows can be entered.	- completed last as the function of the loom was not reliant on the user input.
Fork Movement (Ethan)	- uses the motor encoder to measure the distance the fork travels each time. - after every compression, the fork is moved less than it did prior.	- had to be completed around the same time as the heddle (near the beginning)
Heddle Movement (Josh)	- also uses motor encoder values to measure how far each heddle travels	- had to be completed near the beginning

	- flips the heddles	
Shuttle Passing (Liam and Josh)	- uses two motors on each side of the loom to move the shuttle back and forth - once the shuttle makes contact with the touch sensor, the heddle would flip positions and the fork would compress the strings. - after every pass, the colour sensor would check if there was still string left on the spool. If not, the interface would after the user to replace the spool.	- this had to be the last key function to be completed. Both the heddle and fork compression functions had to be completed prior.
Emergency Stops (Josh)	- if the user presses the left and right buttons on the EV3 Brick, the loom will immediately enter a paused state. - if the user presses the buttons again, the loom will resume exactly from where it left off. - used arrays to store motor values before the pause button was pressed	- the last section of code to be added.

8.0 Conclusion

The loom wove twenty rows, with all sensors, motors, software and pieces working as intended. The constraints that the team set out were not breached, only 4 motors were used, and the middle section of the robot was kept clear for the shuttle to pass. The final design successfully met our desired criteria of having at least 10 rows of warp thread.

Upon demonstration, the robot successfully created several pieces of fabric of various lengths. Several users were able to input their desired number of threads and watch as the robot wove a fabric of that size. All sensors worked as anticipated, including the touch sensors, the motor encoders, and the colour sensor. Furthermore, the emergency pause was demonstrated successfully while the shuttle, heddle, or fork was moving and the thread refill function was triggered by the colour sensor, allowing the team to refill the thread during a particularly long weave.

9.0 Recommendations

9.1 Mechanical Design Recommendations

One of the biggest challenges with this project was creating consistency in the shuttle's path. It frequently derailed, got stuck, or missed the touch sensor. A recommendation would be to use a wider rack for the shuttle since derailing was often caused by the thin laser cut rack leaning over.

Another factor in consistency with this build was that Lego parts are slightly flexible. The team found parts of the structure could bend or move even though they were not supposed to. A recommendation would be to use a Tetrix kit instead of purely Lego.

A limitation of this loom was that it could only weave a certain number of rows, constrained by the depth of the structure. A feature that would improve usability would be a way to store fabric as it is woven to leave space for new rows. This could be achieved by having a roller controlled by an additional motor that rolls up an already woven fabric. This way, the amount of space for new fabric remains constant instead of slowly being filled up.

9.2 Software Design Recommendations

The software was designed such that the user could only input even numbers of rows. This was because the shuttle pass function was written to move the shuttle back and forth. The code could be improved to allow even or odd numbers of rows by using the direction of each pass as a parameter in every function relating to the physical movement.

To make the software easier to read and maintain, all initialization actions could be collected into one module or function. For example, one function moves the shuttle to one side, moves the heddles to the correct position, and receives the user input number of rows. The code for this project performs these as multiple tasks in the main function. Likewise, the shutdown procedures could be collected into one module or function.

Currently, the software requires the fork and heddle to start at specific positions. This is a limitation due to the hardware available, but if the team had access to additional sensors that could find the position of the fork and heddle upon start-up. This would increase the reliability of software and ensure it would work in a greater variety of starting positions.

Back Matter

References

[1] I. Gudrunsdotter, “LOOM,” LUP student Papers, https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8945284&fileOId=8945287 (accessed Dec. 5, 2023).

[2] C. B. Ross, “Understanding weaving: What are looms?,” The Sustainable Fashion Collective, www.the-sustainable-fashion-collective.com/2014/10/14/what-are-looms (accessed Dec. 5, 2023).

Appendix A - Source Code

```
/*
Group # 8-14
Names: Liam Dachner, Ethan Guan, Josh Stadnyk, Alanna Rudolph
Version 5.2
Assumptions: Assuming the string colour is red, the user wants to input an
even amount of rows, and the white heddle plate starts in the upper position.

Sensors and Motors:
      - S4 = touch 1
      - S2 = touch 2
      - S3 = color

      - MotorA = fork
      - MotorB = shuttle 1
      - MotorC = shuttle 2
      - MotorD = heddle
*/

void configureAllSensors();
//configures all sensors to standard configuration

void waitButton(TEV3Buttons button_name);
int getRows();
void flipHeddles(bool cw);
void pushFork(int pass);
void passShuttle(int pass);
void checkEmergencyPause();
void checkSpool();


task main()
{
      configureAllSensors();
      displayString(1 ,  "PRESS ENTER TO START ROBOT:");
      waitButton(buttonEnter);
      //clearScreen();

      int total_passes = getRows();

      time1[T1] = 0;

      motor[motorB] = motor[motorC] = 15;
      while(!SensorValue[S2])
      {checkEmergencyPause();}      //wait until touch sensor is hit
      motor[motorB] = motor[motorC] = 0;

      for (int pass = 0; pass < total_passes; pass++)
      {
```

```
            checkSpool();
            passShuttle(pass);
        }

        eraseDisplay();
        displayString(5, "LOOM COMPLETE!");
        displayString(7, "TIME ELAPSED: %ds", time1[T1]/1000);
        displayString(9, "PRESS ENTER WHEN DONE");
        waitButton(buttonEnter);
}

void configureAllSensors()
{
    SensorType[S1] = sensorEV3_Touch;
    SensorType[S2] = sensorEV3_Touch;
    SensorType[S3] = sensorEV3_Color;
    wait1Msec(50);
    SensorMode[S3] = modeEV3Color_Color;
    wait1Msec(50);
    nMotorEncoder[motorD]=0;
}

void waitButton(TEV3Buttons button_name)
{
    while(!getButtonPress(button_name))
    {}
    while(getButtonPress(button_name))
    {}
}

int getRows()
{
    const int MIN_ROWS = 2;
    const int MAX_ROWS = 20;
    const int INIT_ROWS = 10;
    displayString(3, "Select number of rows");
    int rows = INIT_ROWS;
    displayString( 6, "%d", rows);
    int temp = rows;
    while(!getButtonPress(ENTER_BUTTON))
    {
            if( getButtonPress(buttonUp) )
            {
                  waitButton(buttonUp);
                  rows += 2;
            }

            if( getButtonPress(buttonDown) )
            {
                  waitButton(buttonDown);
```

```
                rows -= 2;
            }

            if( rows<MIN_ROWS || rows>MAX_ROWS )
                rows = temp;

            if( temp != rows )
            {
                displayString(6, "  ");
                displayString( 6, "%d", rows);
                temp = rows;
            }
    }
   return rows/2;
}

void flipHeddles(bool cw)
{
    const float CM_TO_ENC = 360/(2.29*PI);

    if (cw)
    {
            motor[motorD] = 10;
            while (nMotorEncoder[motorD] < 8.5*CM_TO_ENC)
                {checkEmergencyPause();}
            motor[motorD] = 0;
    }
    else
    {
            motor[motorD] = -10;
            while (nMotorEncoder[motorD] > 0)
                {checkEmergencyPause();}
            motor[motorD]=0;
    }
}

void pushFork(int pass)
{
    const float MAX_PUSH_CM = 25;
    const float CM_TO_ENC = 180/(3.91*PI);
    const float DEVIATION = 0.2; //mm

    nMotorEncoder[motorA] = 0;

    motor[motorA] = -25;
    while (abs(nMotorEncoder[motorA]) < (MAX_PUSH_CM*CM_TO_ENC) -
(pass*DEVIATION)*CM_TO_ENC)
    {checkEmergencyPause();}

    motor[motorA] = 0;
```

```
    wait1Msec(500);

    motor[motorA] = 25;
    while (nMotorEncoder[motorA] < 0)
    {checkEmergencyPause();}

    motor[motorA] = 0;
}

void passShuttle(int pass)
{
      //pass forward
      motor[motorB] = motor[motorC] = -15;
      while(!SensorValue[S4])
      {checkEmergencyPause();}      //wait until touch sensor is hit
      motor[motorB] = motor[motorC] = 0;

      flipHeddles(true);
      pushFork(pass);

      motor[motorB] = motor[motorC] = 15;
      while(!SensorValue[S2])
      {checkEmergencyPause();}      //wait until touch sensor is hit
      motor[motorB] = motor[motorC] = 0;

      flipHeddles(false);
      pushFork(pass);

}

void checkEmergencyPause()
{
      if (getButtonPress(buttonLeft) && getButtonPress(buttonRight))
      {
            while (getButtonPress(buttonLeft) || getButtonPress(buttonRight))
            {}
            //create array that stores current motor value, set motors to
zero, resume motors to what they originally are
            int motorValues[4] = { motor[motorA], motor[motorB],
motor[motorC], motor[motorD] };
            motor[motorA] = motor[motorB] = motor[motorC] = motor[motorD] =
0; //immediatley stop everything
            //displayString --> terminated message

            while(!getButtonPress(buttonLeft) ||
!getButtonPress(buttonRight))
            {} //wait for the left and right button to be pressed again
            while(getButtonPress(buttonLeft) || getButtonPress(buttonRight))
            {}
            for (int index = 0; index < 4; index++)
```

```
            {
                  motor[index] = motorValues[index];
            }
      }
}

void checkSpool () {
      if (SensorValue[S3] != (int)colorRed) {
            displayString(9, "please refill spool.");
            displayString(10, "Press enter once refilled");
            while (SensorValue[S3] != (int)colorRed)
                  {}
            waitButton(buttonEnter);
      }
}
```