Faculty of Engineering

Cake Bot Final Report

A report prepared for

The University of Waterloo

Waterloo, Ontario

by

B. Chandrakumar
E. Cho
E. Rasmussen
E. Xie

1A, Mechatronics Engineering

MTE100 and GENE121

Group 37

December 4, 2015

Summary

Cake decoration is a tedious, repetitive and manual process. Cake Bot attempts to improve the process of cake decoration through automation. The solution presented in this design report is a mechatronics system built using Lego NXT parts, Tetrix kits, and other materials such as wood and Plexiglass.

This report documents the design process and creation of Cake Bot. It details the mechanical and software design of the system, analyses of alternative designs, and arrives at the final solution. It also includes conclusions and analyses of the solution presented. This report contains many figures of the mechanical build. Additionally, suggestions are provided to improve both the mechanical and software design of Cake Bot and further development of the project.

This system is not meant to replace the meticulous and artistic process of manual cake decorating, but merely provides an easier alternative for the everyday consumer.

Table of Contents

List of Figures

List of Tables

1.0 Introduction

Decorating cakes is an art form. It takes patience, precision, creativity and superb hand-eye coordination to produce a beautiful and delicious cake. However, producing large quantities of cakes requires an excessive amount of repetitive and manual procedures. While there exists systems to quickly decorate cakes on a mass manufacturing scale, these machines are large and also expensive to develop and manufacture.

Further motivation to create Cake Bot is to apply the knowledge gained throughout the term and to be able to build a mechatronics system, from beginning to end.

1.1 Design Problem Definition

The problem is to improve existing cake decorating processes through a mechatronics system that reduces the amount of repetitive manual input. It is also intended to offer a more compact system that can be used at the consumer scale, since existing oversized automated systems are designed for industrial purposes.

1.2 Goals and Objectives

The goal is to provide an automated alternative to decorating cakes. Specifically, the objective is to decorate a cake quickly and easily, regardless of the level of cake decorating skill the user has. The system is also intended to allow users to be able to customize the design. Customization allows for personal preferences and preset designs through a menu interface.

1.3 Constraints and Criteria

In order for Cake Bot to swiftly and consistently decorate customizable cakes, the design must meet several constraints. One of the constraints is that the system must include a turntable that will allow the cake to spin in a circular motion; this improves application of the icing. The system must also incorporate icing dispensers with wide and narrow nozzles in order to apply the base layer and piping designs on the cake. The system is required to work with a standard cake size of a 9" diameter and 2" height. Cake Bot must include a menu interface for users to choose piping designs. Lastly, none of the Lego NXT components can touch the cake.

Throughout the process of designing and building Cake Bot, there were serval constraints that were modified due to time constraints. Initially, the idea was to include the ability to adjust to any cake size. Throughout the building process the idea became too complex, thus a standard cake size was put in place. For the same reasons as the dynamic cake size, the motion and piping design portion of the vertical system were eliminated. Also, while trying to dispense icing, the prefilled icing bag did not allow for icing to be dispensed evenly for different nozzle types. Hence, icing is directly placed into the container.

An extra functionality added that enhances Cake Bot's ability to decorate cakes is to create different designs based on the colour of the cake placed on the turntable. Given the permitted time period, some other functionalities were proposed but not implemented. These included: the ability to plot more intricate designs, the ability to apply icing between layers of cake, and a mechanism to slice and serve the cake.

The final mechanical design of Cake Bot was judged based on a number of criterion. Ease of access to the nozzles in order to change nozzle type and refill icing is important in our mechanical design. The sturdiness of every component of Cake Bot is also important as Cake Bot is expected to decorate many cakes consistently. Another criteria the design is judged on is simplicity. Since this is a time-sensitive project, it is important that the design is easy to build and quick to modify.

2.0 Mechanical Design

When developing ideas for the design of Cake Bot, there were three main design criteria used to determine which the best concept was: simplicity (easy to build), sturdiness, and ease of access (the ability to change the nozzles and cake easily). Refer to Table 1 for the relative rating of each criterion using a scale of 0.4 to 1.

Table 1: Relative Importance of Criteria

Criterion	Simplicity	Sturdiness	Ease of access	Row Total	Relative Rating
Simplicity	-	0	0	0	0.4
Sturdiness	1	-	0	1	0.7
Ease of access	1	1	-	2	1

Each design was rated good (holding a value of 1), okay (value of 0.5), or bad (value of 0.2) for each of the criteria. Refer to Appendix C for the initial design drawings.

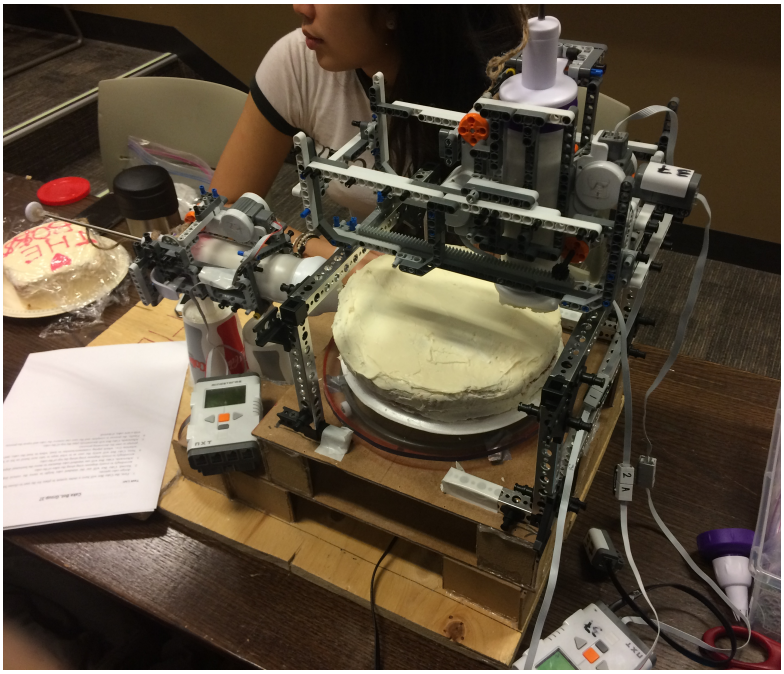


Figure 1: Final Cake Bot Prototype

As seen in Figure 3, this is the final design of Cake Bot, but before this occurred, many designs were considered.

2.1 Turntable

Circular motion was required to evenly apply icing to the cake, which was accomplished through a turntable mechanism. The circular motion allowed for the top and sides of the cake to be iced, as well as the creation of decorative piping patterns. In the idea development stage of the turntable, three designs were suggested and analyzed based on the main design criteria. Refer to Table 2 for the turntable design ratings.

Table 2: Design Ratings for the Turntable Mechanism

Criterion	Simplicity	Sturdiness	Ease of access	Design Rating
Design A: Connect motor direct to turntable shaft	Good	Okay	Good	1.75
Design B: Gear motor to turntable shaft	Okay	Good	Bad	1.1
Design C: Motor to shaft through a pulley system	Bad	Okay	Bad	0.63

Design A was very easy to build and easily accessible since a minimum number of parts are used. However, design B was a more robust solution and provides more freedom for controlling the motor speed. Design B also required additional parts, which made construction more complex and modifications more difficult. Design C was also quite complex to build and access, and was less sturdy due to potential slippage. Design A was ranked the highest in terms of the proposed criteria. Refer to Appendix E for design rating calculations.

Initially, the turntable design decided upon was to connect the shaft of the turntable directly to a motor secured within the base support structure. There were not any major changes to the overall turntable design, but there were some minor changes throughout the building process. For example, there was no way to easily make a turntable plate out of Lego, thus a circular turntable plate was cut from a sheet of ¼" thick plexiglass. Another change made to the design was to add a circular bearing frame taken from a microwave. This was added to reduce friction by lifting the turntable plate off the wooden base support structure, allowing the turntable to glide easily and ensure consistent motion. However, if the circular bearing frame was secured directly onto the base, the wheels on the frame would rub against it. Thus, the frame was raised off the base with Lego pieces, and could spin freely.

Once the turntable was fully built, the final design was much like a record player. The motor was mounted on the main base under the turntable, and was directly connected with a shaft. A circular bearing frame with wheels was mounted beneath the turntable plate to reduce friction between the base and turntable plate.

2.2 Dispenser Movement Mechanism

Horizontal and vertical motion were required to ice and decorate the entire cake surface. These motions combined with the circular motion of the turntable allowed Cake Bot to access any point on the cake. Two dispenser movement designs were formulated and judged based on the design criteria in the idea development stage. Refer to Table 3 for the dispenser movement design ratings.

Table 3: Design Ratings for the Dispenser Movement Mechanism

Criterion	Simplicity	Sturdiness	Ease of access	Design Rating
Design A: Combine horizontal and vertical motion by curved rack and pinions	Good	Bad	Bad	0.74
Design B: Separate horizontal and vertical motion	Okay	Good	Okay	1.3

A proposed idea for moving the icing dispenser(s) was design A: one dispenser travels along a curved rack and pinion. This design, while simple, required special parts not provided in the kits, such as the curved racks that required 3-D printing. This design was also weak and not robust. A simpler design was design B, which separates the top and side dispensers. Movement of the top and side dispensers was easily achieved with rack and pinion gears provided. This design was also fairly sturdy and easy to modify. Design B was ranked the highest based on the proposed criteria. Refer to Appendix E for design rating calculations.

The initial decision for the icing dispenser motion was to have to different systems for the horizontal and vertical motion. As building commenced, the vertical motion component became very difficult to manipulate. To achieve vertical motion, the idea was to use hydraulics to push the vertical icing dispenser up and down through a motor using racks and pinions. The force required to push the hydraulics was too large for the motor to handle thus, causing the pinion to slip. Due to time constraints, the vertical motion was eliminated. In order to ice the sides of the cake, the wide nozzle length of two inches was taken into consideration, and so, the cake was decided to be two inches tall. In order to incorporate both nozzles, the dispenser would have to move. Since motion was no longer part of the design, the decision was made to not decoratively pipe the sides of the cake.

Due to parts constraints, the initial design for the horizontal motion was to have racks on one side of the frame. This idea was quickly discovered to be unrealistic as all the weight of the icing dispenser would rely on one pinion to move and this increased the chances of gear slippage. The modification to the design was to have racks on both sides of the dispenser to evenly distribute to weight. This design required 12 racks to execute the process thus, an external source provided those racks.

The final build of the horizontal structure looked similar to a bridge. There were racks along the sides of the bridge and the icing dispenser sat on top of them. This design allowed the horizontal icing dispenser to move back and forth on the bridge in order to ice along the radius of the cake through the use of racks and pinions.

2.3 Icing Dispenser

In order to release icing from the dispenser, a mechanism that had the ability of travelling in linear motion was required and so, a mechanism that could pull the icing dispenser shaft in was needed. Refer to Table 4 for the icing dispenser mechanism design ratings.

Table 4: Design ratings for dispensing icing mechanism

Criterion	Simplicity	Sturdiness	Ease of access	Design Rating
Design A: With a piece of string attaching motor and top of dispenser	Good	Okay	Good	1.75
Design B: Hydraulics	Bad	Okay	Okay	0.93

Design A fit many of the proposed criterion. Since a string controlled the dispensing action, it was very easy to build and remove for icing refills. Design A was also light and relatively sturdy. Design B required a complex support system in order to hold the hydraulics in place, especially since the icing dispenser shaft extended quite far from the container. It would also be difficult for the user to remove the dispenser to refill. Design A was ranked the highest based on the proposed criteria. Refer to Appendix E for design rating calculations.

The initial design for the icing dispenser involved a string connected to the end of the extended dispenser shaft and to a motor axle acting as a spool. As the motor spun, the string would wrap around the spool and pull the shaft in thus, dispensing the icing. There were not any design changes to the mechanism that controlled the dispenser, but the structure that held the container did undergo changes. Modification were made to the frame to allow for the removal and locking in of the dispenser. This involved a Lego structure around the dispenser

handle where turning the handle locked and unlocked the dispenser, allowing for security and easy of access. Another change made was adding a removable axle to the frame to allow the dispenser to be locked in further in or out of the frame to accommodate for the different piping nozzles heights.

In summary, the final design of the icing dispenser consists of a fishing rod method of pulling the dispenser shaft in. There is also a locking mechanism on the dispenser frame to hold the dispenser securely in place while giving the user the ability to easily remove the dispenser. Finally, the frame is designed to accommodate for different nozzle heights.

2.4 Support Structure

Cake Bot's support structure was designed based off of the chosen designs for the major components. Most of the design decisions for the support structure were put forward during the building stage as there were components of the structure that were not considered until the other systems were completed.

Initially, the support structure consisted of three main sub-structures: the base, the horizontal, and the vertical dispenser components. The base was designed to house the entire robot. The horizontal and vertical support had to place the dispensers a certain distance above and away from the cake while maintaining the sturdiness of the components.

The initial base structure consisted of a main base with a box to house the turntable. Due to the hydraulics, the turntable had to be raised 2.5 inches from its original position hence, another box structure was built. The structure to hold the horizontal component has to be sturdy to keep everything in place and so, the design involved many structural beams. This initial design did not accommodate for the need of inserting and removing a cake and so, the structural integrity of the frame was traded for ease of access. The vertical motion component was initially planned to be a part of the horizontal structure, but the framed had to be moved away to be able to ice the sides of the cake. Therefore, the vertical structure was made from paper cups and syringes, as vertical motion was still in mind during this part of the build. Though the vertical motion was taken out of the design, the structure stayed as it fulfilled the requirements for that component.

Our final design consists of a main base of plywood with two box particle board structures to hold the turntable. Along the side of the base, there are two U-shaped structures going across the cake to house the horizontal motion component. Off to the side is the vertical structure that keeps the vertical icing dispenser in place.

3.0 Software Design

The finalized software design for Cake Bot is organized into 3 main segments: the user interface, the top decoration processes, and the side decoration processes. Since Cake Bot's design requires four motors, the code runs using two different NXT Bricks and is split into two software files that run independent of one another. One NXT Brick controls the turntable and the horizontal icing dispenser motions while the other NXT Brick controls the vertical icing dispenser motions. Cake Bot uses 4 sensors: NXT buttons, a touch sensor, motor encoders, and a colour sensor. Refer to Appendix A for the source code.

On a high level, the turntable and horizontal icing dispenser software runs according to a series of steps in order to ice the top of the cake. First, the main menu is displayed where Cake Bot welcomes the user. Cake Bot then calibrates its horizontal position using a touch sensor and the initial icing dispensing position using relative encoder positioning. Next, a menu system is displayed that allows the user to select from a variety of cake design choices, or for the cake design to be based on the cake colour. Then Cake Bot pipes a basecoat of icing onto the cake and prompts the user on which icing colour and piping nozzle to load next. Cake Bot then pipes the selected design onto the cake. Finally, Cake Bot resets the motors to the initial positions, displays decorating time in seconds, and prompts the user to end the program or ice another cake. Refer to Figure 2 the top icing procedure flowchart.

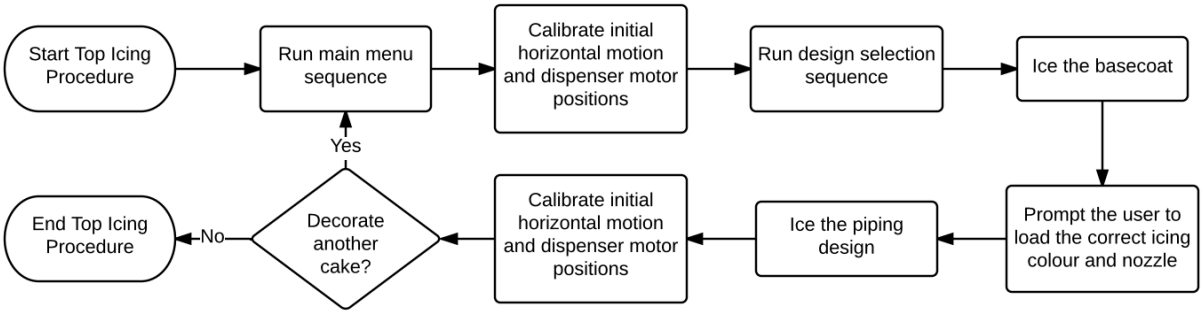


Figure 1: Top Icing Procedure Flowchart

The vertical icing dispenser software runs in a similar fashion. It uses a series of menu systems and motor motions in order to decorate the sides of the cake. Refer to Figure 3 for a flowchart of the side icing procedure.

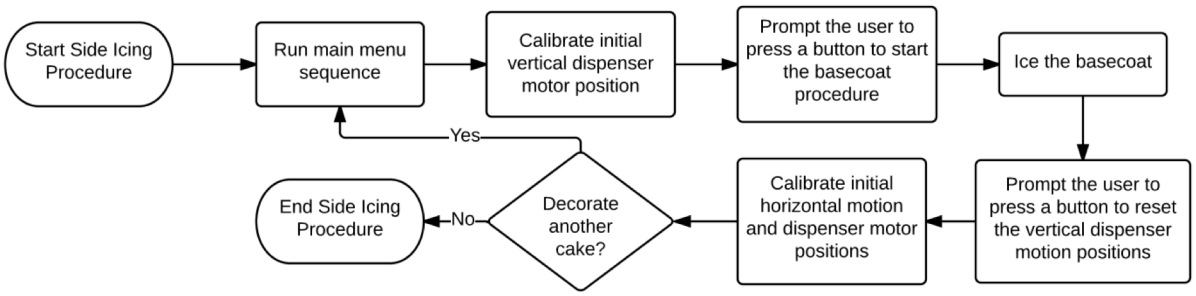


Figure 2: Side Icing Procedure Flowchart

Each software file contains similar functions: waitForButtonPress, setMotor, brakeMotor, moveDistance, resetPositions, and menu. The waitForButtonPress function handles the detection of a specific NXT button or any NXT button being pressed then released. The setMotor function handles the setting of speed and direction of any motor in an intuitive manner. Instead of remembering which motor belongs to which port and what direction each motor has to travel, setMotor takes in enumerated types that describe the intent. For example, "turntable" is used instead of "motorA". The brakeMotor function brakes the motor instead of letting the motor coast to zero, allowing for more accurate positioning. The moveDistance function was created for similar reasons to the setMotor function, but instead of taking in only speed and direction, it also takes in a requested encoder distance. The resetPositions function runs before and after each decorating procedure to bring the motors back to a known position. Lastly, the menu function was created to segment the different portions of the menu system in order to make the code easier to read.

The horizontal icing software file includes a few more functions than the vertical file, such as waitTurnTable, getColorType, and decorateTop. The waitTurnTable function is used to time the speed and position of the horizontal dispenser motion based on the encoder degrees of the turntable. The getColourType function reads the colour sensor and returns a design selection based on the colour. The decorateTop function runs the requested autonomous decorating sequence, which could be one of four functions: iceBasecoat, iceSpiralPipe, iceDottedEdgePipe, or iceContinuousEdgePipe. These four functions run combinations of the functions described above to create piping designs.

Initial tests consisted of testing the mechanical systems and sensor limits. Then the resetPositions and setMotor functions were created and tested using the mechanical system. Next, the moveDistance and brakeMotor functions were created since there were repeated patterns within the resetPositions function where motors were running for an encoder distance. While these functions were being

developed, a basic version of the menu system was created to allow for testing of the autonomous icing sequences. Once the menu was completed, the autonomous icing sequences for the horizontal motions, along with the waitTurnTable function, were tested using a marker on a cake tin. Next, the full version of the menu was developed. After this point, the horizontal icing code was complete, and was tested many times using whipped cream frosting and a real cake to further refine the autonomous icing sequences. This testing process was repeated for the vertical icing code.

The main issue encountered while developing the software was developing a method of controlling four motors. Initially, the plan was to use a motor multiplexer to control the horizontal and vertical icing motions with two tasks that could run simultaneously. However, a motor multiplexer requires a 9V battery and no 9V batteries could be found in reasonable time to complete testing. The next course of action was to link two NXT Bricks through Bluetooth. Due to time constraints, however, this plan proved to be difficult. So, the selected method of controlling four motors was to have two NXT Bricks run different programs. The user is prompted by the menu system to run the vertical and horizontal icing procedures at the appropriate time. This method of running the motors greatly changed from the previously presented structure of the program. Refer to Appendix D for a flowchart of the previous program structure.

4.0 Project Management

Each group member has their own strengths that the team strived to use by splitting tasks amongst group members accordingly. For instance, Ella focused on the component portion of the build, Biranavi focused on the housing and platform portion of the build, and Emily and Emma focused on assembling the support structure using the Tetrix pieces. All members of the team participated in creating the software and the functions were delegated as indicated in the source code available in Appendix A.

Figure 4 shows the project timeline in a chart based on William Melek's teachings [1].

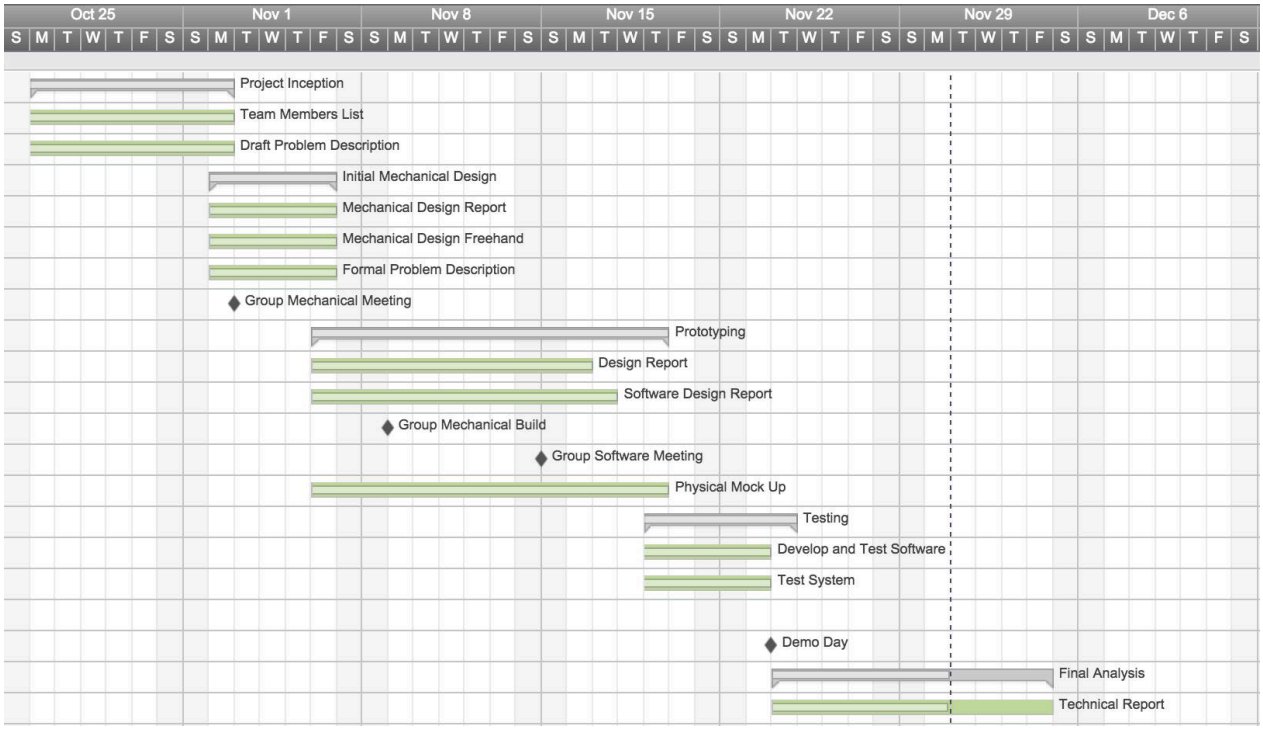


Figure 4: Gantt Chart

As shown in the above chart, the project timeline was split into four major parts. This included the idea inception, mechanical build, software development, and testing. The idea stage involved establishing team roles and drafting the design problem. The next stage involved drawing up the different mechanical designs. From here, the physical build was created and programming commenced. Lastly the individual parts of the software were unit tested and tested as a whole.

From the original project timeline, the schedule changed due to the complexity of the mechanical design. The physical build was extended by two days, and this caused the software development and testing to be pushed forward by two days.

5.0 Conclusions

Cake Bot was able to meet many of the constraints defined throughout the project. Cake Bot is able to decorate a cake of a standard size by using a turntable, a horizontal icing dispenser that moves along the radius of the cake, a vertical stationary icing dispenser, and appropriate support. All components of Cake Bot moved well together and successfully applied icing in three different designs on the top of the cake, as proposed.

Judging the final design based on the defined criterion, the system was relatively sturdy and the icing dispensers were easily accessible. Cake Bot was not able to accommodate for variations outside of a perfectly circular cake shape, so the icing outcome was not always as desired. However, it did fulfill its original goals and objectives of decorating a cake quickly and easily while allowing for user customization.

Overall, Cake Bot was a great application of the knowledge gained in the MTE 100 and GENE 121 courses this term.

6.0 Recommendations

A mechanical change that could be made to Cake Bot to improve its design is modifying the hydraulics that were intended to lift the vertical icing dispenser. One of the issues encountered while constructing the robot was that the hydraulics required more force than the NXT motor could provide. To improve the hydraulic system, the lengths and volumes of the syringes and tubing can be changed. This would give mechanical advantage to the motor and make the system much easier for the motor to lift.

Another possible mechanical design change would be to replace the plastic Lego shaft used to move the turntable with a less flexible material, such as wood, metal, or a harder form of plastic. Because of the flexibility of the axle, when the motor accelerates and decelerates, the turntable lags behind. This causes the motion of the turntable to be inconsistent for the first few rotations. Due to this inconsistency, Cake Bot is programmed to wait for the turntable to get up to speed and stabilize. This makes it difficult to create designs that require the turntable to move small distances. By using a stiffer material for the axle, this issue would be solved.

On the software side, there were issues with controlling four motors with a common source code. The motor multiplexer could have been used to control four motors with one NXT Brick, but no 9V batteries were accessible during the testing phase. The backup plan was to run the four motors off of two different NXT Bricks. This solution was acceptable, considering the time constraints. However, a more robust solution would have been to use Bluetooth communication between the two NXT Bricks or have access to a 9V battery for the motor multiplexer.

7.0 References

[1] W. Melek, MTE 100. Class Lecture, Topic: "Project Planning", University of Waterloo, 2015.

Appendix A - Source Code

Horizontal Cake Decorating Code: CakeBots_Horiz.c

```
// --- Sensor Ports
#define SV_Color        SensorValue[S4]
#define SV_TouchStart   SensorValue[S3]
#define SV_TouchEnd     SensorValue[S2]

// --- Horizontal Motion Positions (in encoder degrees)
#define H_DIST                  802
#define H_BASECOATSTART         85
#define H_BIGNOZZLE             460

// --- enumerate motor ports
enum tMotorType
{
      turnTable,
      H_motion,
      H_icing
};

// --- enumerate motion directions
enum tDirection
{
      forward,
      backward,
      in, // icing in
      out // icing out
};

// --- enumerate piping procedures
enum tDecorateType
{
      basecoat,
      spiralPipe,
      dottedEdgePipe,
      continuousEdgePipe,
      none
};

// --- enumerate menus
enum tMenuSection
{
      mainMenu,
      selectDesign,
```

```
	userPrompt,
	backToMain
};

// --- Biranavi wrote this function
void waitForButtonPress(int button = -1)
{
	// --- if no specific button is selected, check for any button
	if(button == -1)
	{
		while(nNxtButtonPressed == -1){wait1Msec(50);}
		wait1Msec(200);
		while(nNxtButtonPressed != -1){wait1Msec(50);}
		wait1Msec(200);
	}
	else
	{
		while(nNxtButtonPressed != button){wait1Msec(50);}
		wait1Msec(200);
		while(nNxtButtonPressed == button){wait1Msec(50);}
		wait1Msec(200);
	}
}

// --- Ella wrote this function
void setMotor(tMotorType motorType, int speed, tDirection dir =
forward)
{
	// --- set motor speeds based on direction and motor type
	switch(motorType)
	{
	case H_motion:
		if(dir == forward)
			motor[H_motion] = speed;
		else
			motor[H_motion] = -speed;
		break;

	case H_icing:
		if(dir == out)
			motor[H_icing] = -speed;
		else
			motor[H_icing] = speed;
		break;

	case turnTable:
```

```
            if(dir == out)
                  motor[turnTable] = -speed;
            else
                  motor[turnTable] = speed;
            break;

      default:
            break;
      }
}

// --- Ella wrote this function
void brakeMotor(tMotorType motorType)
{
      // --- set motor to a small power in opposite direction to brake
      setMotor(motorType, 5 * -1 * sgn(motor[motorType]));
      wait1Msec(200);
      setMotor(motorType,0);
}

// --- Ella wrote this function
void moveDistance(tMotorType motorType, int speed, tDirection dir, int
distance, bool stopMotor = true)
{
      setMotor(motorType, speed, dir);

      // --- wait for encoder to reach requested position
      if(dir == backward || dir == out)
            while( nMotorEncoder[motorType] > -distance )
            {wait1Msec(50);}
      else
            while(nMotorEncoder[motorType] < distance){wait1Msec(50);}

      if(stopMotor)
            brakeMotor(motorType);
}

// --- Emma wrote this function
void resetPositions(bool startPos = true)
{
      // set turnTable motion to 0
      setMotor(turnTable, 0);

      // reset horizontal motion
      setMotor(H_motion, 50, forward);
      while(SV_TouchStart == 0){}
```

```
	setMotor(H_motion, 0);
	nMotorEncoder[H_motion] = 0;

	// reset horizontal icing dispencing motion
	if(startPos)
		moveDistance(H_icing, 70, out, 1400);
	else
		moveDistance(H_icing, 70, in, 0);
}

// --- Ella wrote this function
void waitTurnTable(int degrees, bool reset = true)
{
	if(reset)
		nMotorEncoder[turnTable] = 0;

	// --- wait for turntable to reach requested position
	while(nMotorEncoder[turnTable] < degrees){wait1Msec(50);}
}

// --- Ella wrote this function
tDecorateType getColorType(TColors color)
{
	// --- detect the colour of the cake and select a design type
	if(color == BLUECOLOR)
		return spiralPipe;
	else if(color == REDCOLOR)
		return dottedEdgePipe;
	else
		return continuousEdgePipe;
}

// -------------------------------- ICING PRODECURE FUNCTIONS (START)

// --- Ella wrote this function
void iceBasecoat()
{
	// get to starting position and let turntable get up to a
consistant speed
	setMotor(turnTable, 20, forward);
	waitTurnTable(1080);

	setMotor(H_icing, 45, out);
	waitTurnTable(580);

	setMotor(H_icing, 0);
```

```
	moveDistance(H_motion, 70, backward, H_BIGNOZZLE);

	setMotor(H_icing, 20, out);
	waitTurnTable(400);

	brakeMotor(H_icing);
	brakeMotor(turnTable);
}

// --- Biranavi wrote this function
void iceSpiralPipe()
{
	// let turntable get up to a consistent speed
	setMotor(turnTable, 30, forward);
	waitTurnTable( 1080 );

	setMotor(H_icing, 35, out);
	wait1Msec(800);
	moveDistance(H_motion, 20, backward, H_BASECOATSTART +
H_BIGNOZZLE);

	brakeMotor(H_motion);
	brakeMotor(H_icing);
	brakeMotor(turnTable);
}

// --- Emma wrote this function
void iceDottedEdgePipe()
{
	// let turntable get up to a consistant speed
	moveDistance(H_motion, 20, backward, 100);

	for(int i = 0; i < 6; i++)
	{
		setMotor(turnTable, 40, forward);
		waitTurnTable(60);
		setMotor(turnTable, 0);
		setMotor(H_icing, 80, out);
		wait1Msec(450);
		brakeMotor(H_icing);
	}

	brakeMotor(H_icing);
	brakeMotor(turnTable);
}
```

```
// --- Emily wrote this function
void iceContinuousEdgePipe()
{
	// let turntable get up to a consistant speed
	setMotor(turnTable, 20, forward);
	waitTurnTable(1080);

	// make a dot every 20 degrees of the turntable rotation
	setMotor(H_icing, 60, out);
	waitTurnTable(400);

	brakeMotor(H_icing);
	brakeMotor(turnTable);
}
// ---------------------------------- ICING PRODECURE FUNCTIONS (END)

// --- Ella wrote this function
void decorateTop(tDecorateType type)
{
	// --- run the requested autonomous decoration function
	switch(type)
	{
	case basecoat:
		iceBasecoat();
		break;

	case spiralPipe:
		iceSpiralPipe();
		break;

	case dottedEdgePipe:
		iceDottedEdgePipe();
		break;

	case continuousEdgePipe:
		iceContinuousEdgePipe();
		break;

	default:
		break;
	}
}

// ---------------------------------------------- MENU SYSTEM (START)

// --- Ella and Emily wrote this function
```

```
void menu(tMenuSection menuSelection, tDecorateType pipe = none)
{
	switch(menuSelection)
	{
	case mainMenu:
		// make sure all buttons are released
		wait1Msec(200);
		while(nNxtButtonPressed != -1){}
		wait1Msec(200);

		nMotorEncoder[H_motion] = 0;
		nMotorEncoder[motorB] = 0;
		nMotorEncoder[turnTable] = 0;

		displayString(0, "WELCOME");
		displayString(2, "Press <ENTER> to");
		displayString(3, "start");
		waitForButtonPress();

		break;

	case selectDesign:
		// make sure all buttons are released
		while(nNxtButtonPressed != -1){}

		eraseDisplay();
		displayString(0, "Customized?");
		displayString(3, "<: Yes");
		displayString(4, ">: No");

		while (nNxtButtonPressed == -1){}
		if(nNxtButtonPressed == 1)
		{
			wait1Msec(200);
			while( nNxtButtonPressed != -1 ){}
			wait1Msec(200);

			eraseDisplay();
			displayString(0, "Press <ENTER> to");
			displayString(1, "detect color");
			waitForButtonPress();
			pipe = getColorType(SV_Color);
		}
		else
		{
			wait1Msec(200);
```

```
            while(nNxtButtonPressed != -1){}
            wait1Msec(200);

            eraseDisplay();
            displayString(0, "Select Design");
            displayString(4, "<: Dotted");
            displayString(3, ">: Continuous");
            displayString(5, "<ENTER>: Spiral");

            while (nNxtButtonPressed == -1){}
            if(nNxtButtonPressed==1)
                  pipe= continuousEdgePipe;
            else if (nNxtButtonPressed == 2)
                  pipe= dottedEdgePipe;
            else
                  pipe= spiralPipe;
      }

      wait1Msec(200);
      while(nNxtButtonPressed != -1){}
      wait1Msec(200);

      eraseDisplay();
      displayString(0, "Press <ENTER> to");
      displayString(1, "begin icing");
      waitForButtonPress();
      eraseDisplay();
      break;

case userPrompt:
      eraseDisplay();
      if(pipe == spiralPipe)
      {
            displayString(2, "Insert Nozzle 1");
            displayString(3, "and Red Icing");
      }
      else if(pipe == dottedEdgePipe)
      {
            displayString(2, "Insert Nozzle 2");
            displayString(3, "and Blue Icing");
      }
      else
      {
            displayString(2, "Insert Nozzle 3");
            displayString(3, "and Purple Icing");
      }
```

```
            displayString(6, "Press <ENTER>");
            waitForButtonPress();
            eraseDisplay();
            break;

      case backToMain:
            eraseDisplay();
            // display time it took to decorate the cake
            displayString(0, "Decorate Time %ds", time1[T1] / 1000);
            displayString(1, "Press <ENTER> to");
            displayString(2, "ice another cake");

            displayString(4, "Press <ESC> to");
            displayString(5, "exit");
            while( nNxtButtonPressed == -1 ){}
            eraseDisplay();
            break;
      }
}

// ------------------------------------------------ MENU SYSTEM (END)

// --- Ella wrote this task
task main()
{
      tDecorateType pipeType;

      SensorType[S4] = sensorColorNxtFULL;
      SensorType[S3] = sensorTouch;

      while(nNxtButtonPressed != 0)
      {
            time1[T1] = 0;

            menu(mainMenu);
            resetPositions();

            menu(selectDesign, pipeType);

            decorateTop(basecoat);
            resetPositions(false);

            menu(userPrompt, pipeType);
            resetPositions();
```

```
            decorateTop(pipeType);
            resetPositions(false);

            menu(backToMain);
      }
}
```

Vertical Cake Decorating Code: CakeBots_Vert.c

```
const int  V_icing = 0;

enum tDirection
{
      in, // icing in
      out // icing out
};

enum tMenuSection
{
      mainMenu,
      waitToStart,
      resetDispenser,
      backToMain
};

// --- Bianavi wrote this function
void waitForButtonPress(int button = -1)
{
      // --- if no specific button is selected, check for any button
      if(button == -1)
      {
            while(nNxtButtonPressed == -1){wait1Msec(50);}
            wait1Msec(200);
            while(nNxtButtonPressed != -1){wait1Msec(50);}
            wait1Msec(200);
      }
      else
      {
            while(nNxtButtonPressed != button){wait1Msec(50);}
            wait1Msec(200);
            while(nNxtButtonPressed == button){wait1Msec(50);}
            wait1Msec(200);
      }
}
```

```
// --- Ella wrote this function
void setMotor(int speed, tDirection dir = out)
{
	// --- set motor speeds based on direction and motor type
	if(dir == out)
		motor[V_icing] = -speed;
	else
		motor[V_icing] = speed;
}

// --- Ella wrote this function
void brakeMotor(int motorType)
{
	// --- set motor to a small power in opposite direction to brake
	setMotor(motorType, 5 * -1 * sgn(motor[motorType]));
	wait1Msec(200);
	setMotor(motorType, 0);
}

// --- Ella wrote this function
void moveDistance(int motorType, int speed, tDirection dir, int
distance, bool stopMotor = true)
{
	setMotor(speed, dir);

	// --- wait for encoder to reach requested position
	if(dir == out)
		while(nMotorEncoder[motorType] > -distance)
		{wait1Msec(50);}
	else
		while(nMotorEncoder[motorType] < distance){wait1Msec(50);}

	if(stopMotor)
		brakeMotor(motorType);
}


// --- Ella wrote this function
void resetPositions(bool startPos = true)
{
	// reset vertical icing dispencing motion
	if(startPos)
		moveDistance(V_icing, 70, out, 1200);
	else
		moveDistance(V_icing, 70, in, 0);
}
```

```
// --------------------------------- ICING PRODECURE FUNCTIONS (START)

// --- Ella wrote this function
void iceBasecoat()
{
	// --- Note to remember: vertical and horizontal components are
icing at the same time
	wait1Msec(1000);

	setMotor(40, out);
	wait1Msec(4000);

	brakeMotor(V_icing);
}
// ---------------------------------- ICING PRODECURE FUNCTIONS (END)

// ------------------------------------------------ MENU SYSTEM (START)

// --- Ella wrote this task
void menu(tMenuSection menuSelection)
{
	switch(menuSelection)
	{
	case mainMenu:
		wait1Msec(200);
		while(nNxtButtonPressed != -1){}
		wait1Msec(200);

		nMotorEncoder[V_icing] = 0;

		displayString(0, "WELCOME");
		displayString(2, "Press <ENTER> to");
		displayString(3, "start");
		waitForButtonPress();
		break;

	case waitToStart:
		eraseDisplay();
		displayString(0, "Press <ENTER> to");
		displayString(1, "begin icing");
		waitForButtonPress();
		eraseDisplay();
		break;

	case resetDispenser:
```

```
            eraseDisplay();
            displayString(0, "Press <ENTER> to");
            displayString(1, "reset");
            waitForButtonPress();
            eraseDisplay();
            break;

      case backToMain:
            eraseDisplay();
            displayString(0, "Press <ENTER> to");
            displayString(1, "ice another cake");

            displayString(4, "Press <ESC> to");
            displayString(5, "exit");
            while( nNxtButtonPressed == -1 ){}
            eraseDisplay();
            break;
      }
}

// ------------------------------------------------ MENU SYSTEM (END)

task main()
{
      while(nNxtButtonPressed != 0)
      {
            menu(mainMenu);
            resetPositions();

            menu(waitToStart);
            iceBasecoat();

            menu(resetDispenser);
            resetPositions(false);

            menu(backToMain);
      }
}
```

Appendix B - Robot Photos

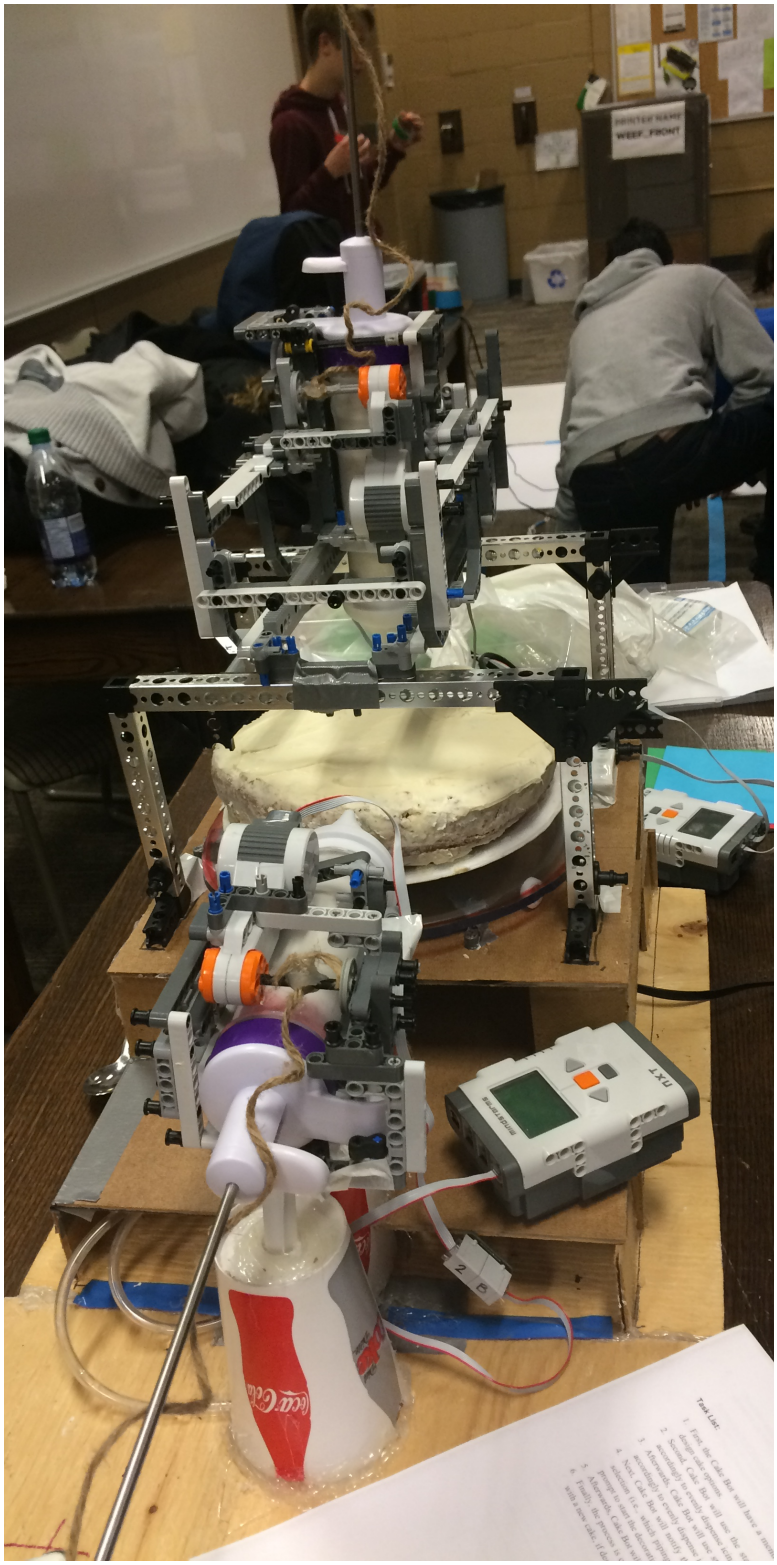


Figure: B.1: Full Cake Bot Back View

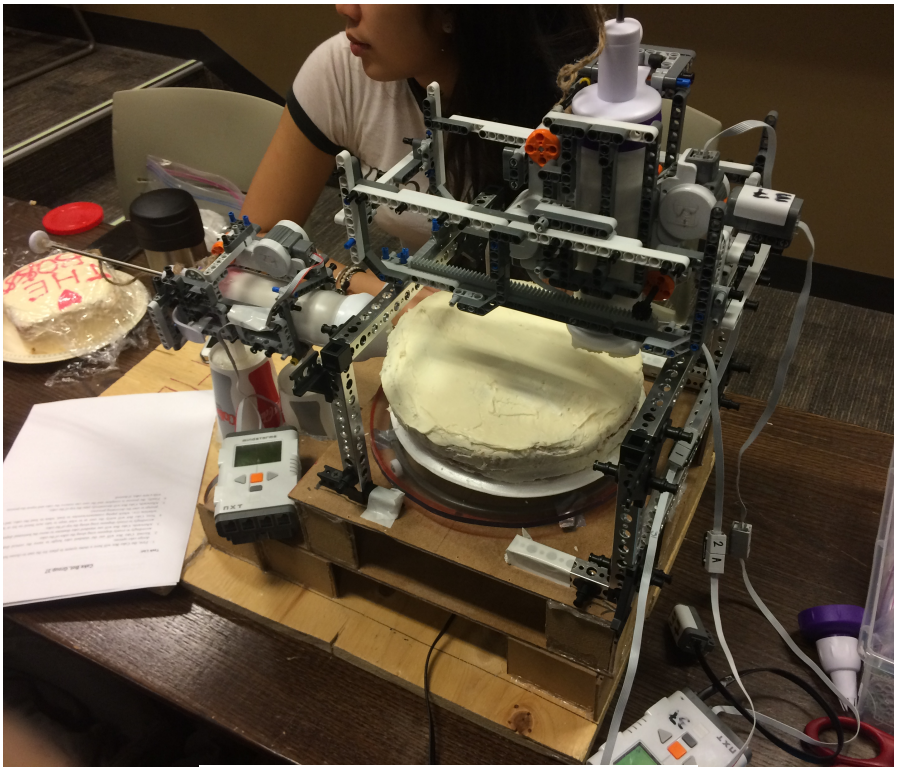


Figure: B.2: Full Cake Bot Side View

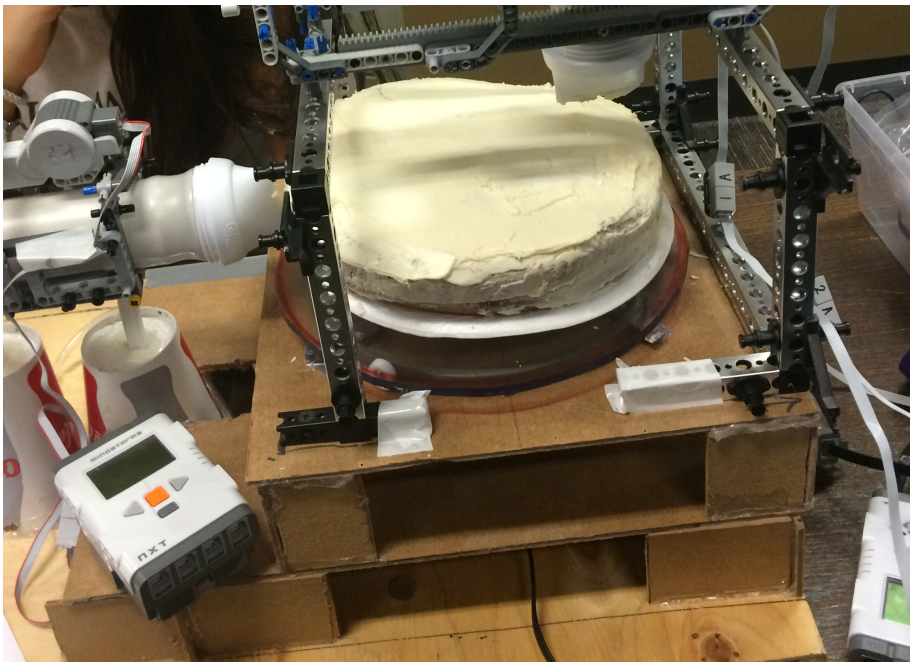


Figure: B.3: Cake Bot Turntable

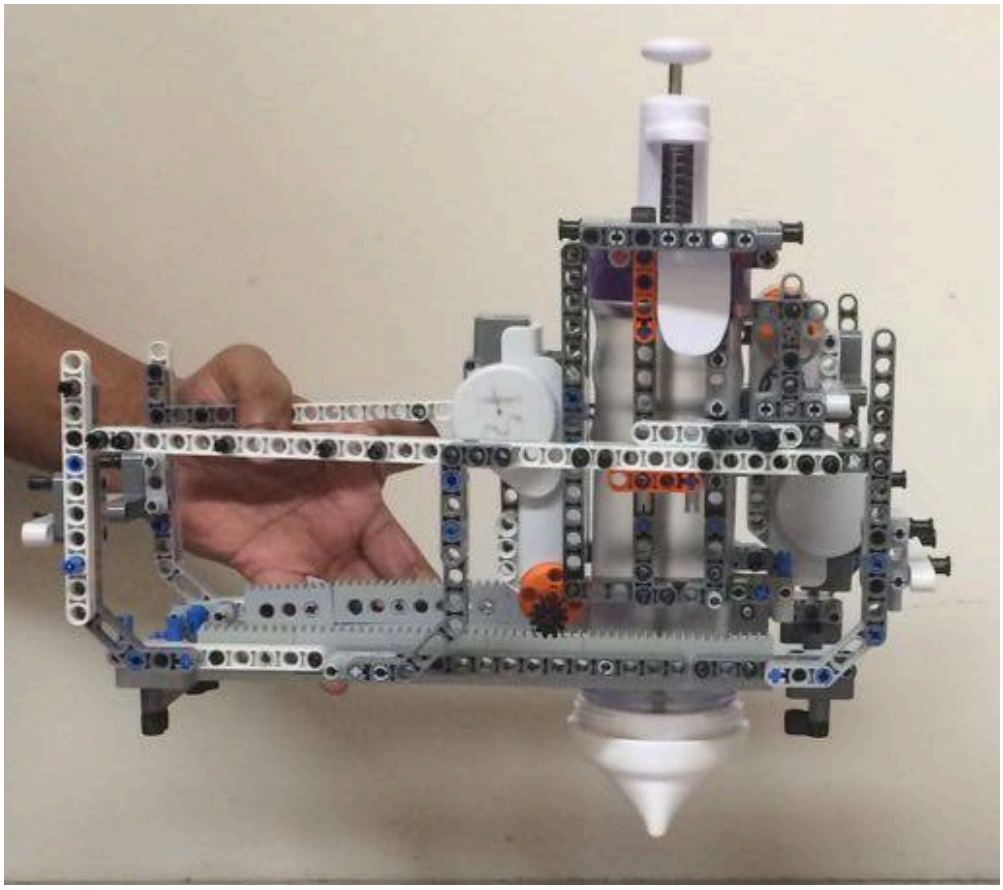


Figure: B.4: Cake Bot Horizontal Icing Dispenser

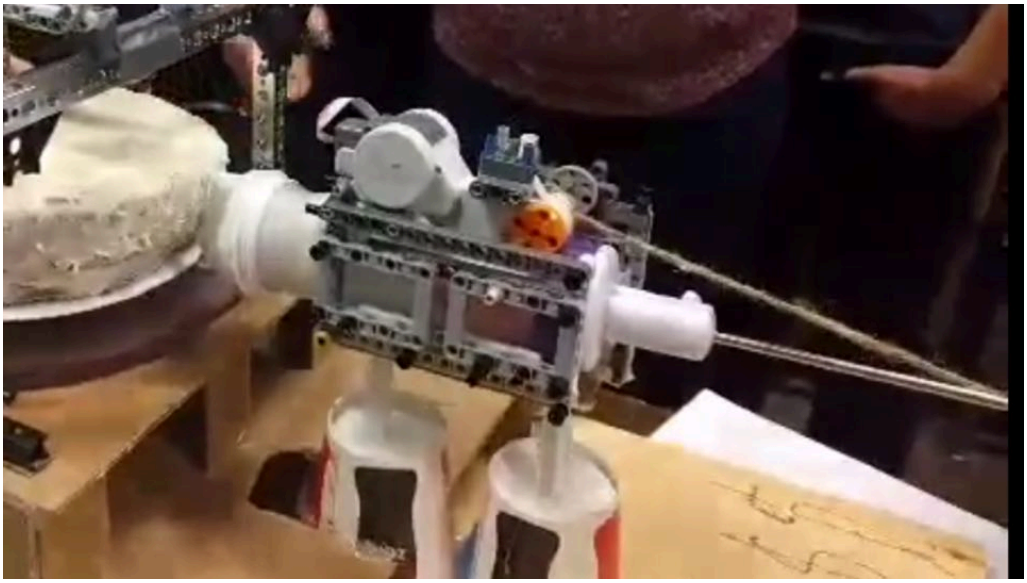


Figure: B.5: Cake Bot Vertical Icing Dispenser

Appendix C - Alternative Mechanical Design Drawings

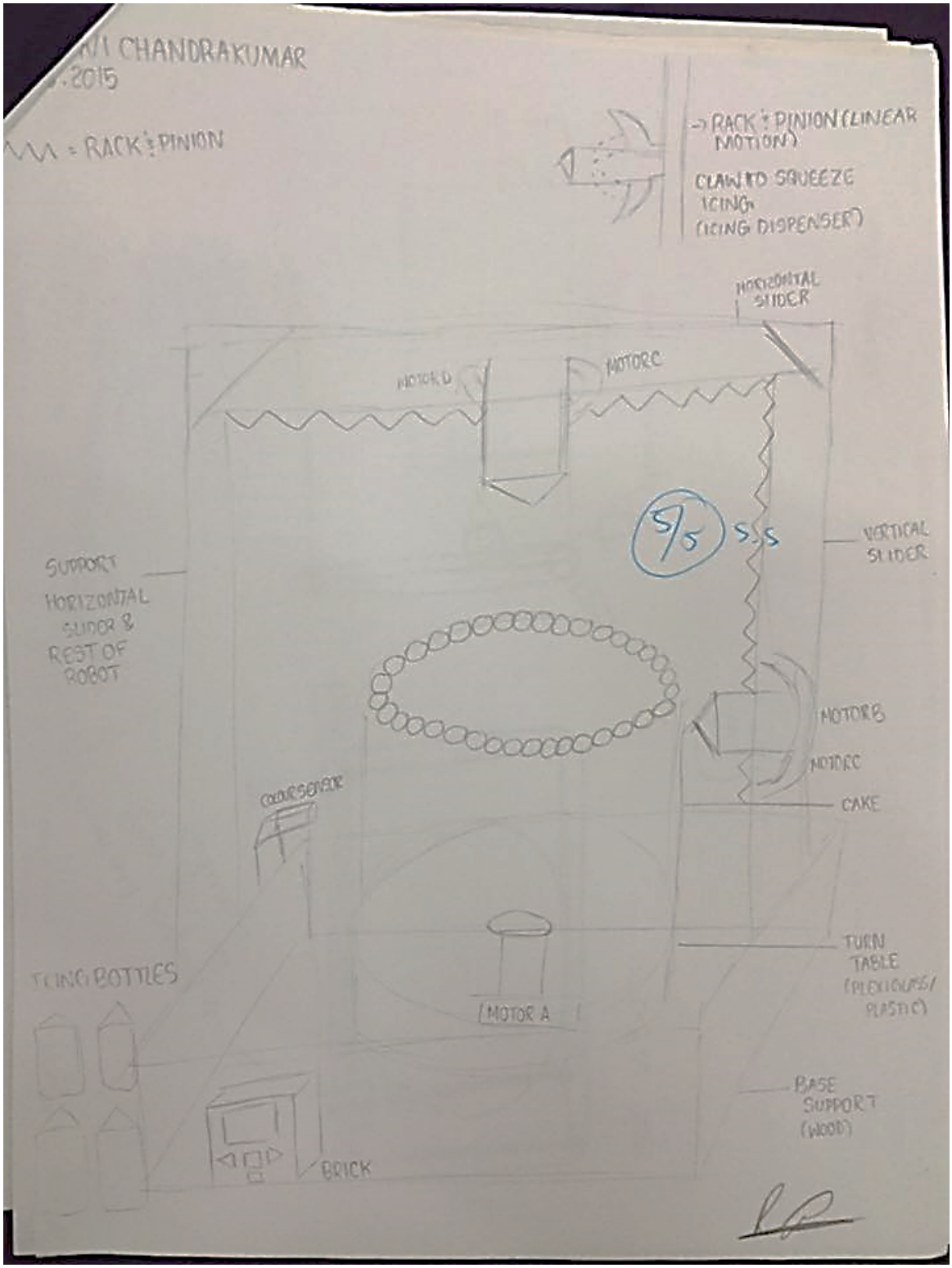


Figure: C.1: Cake Bot Concept 1

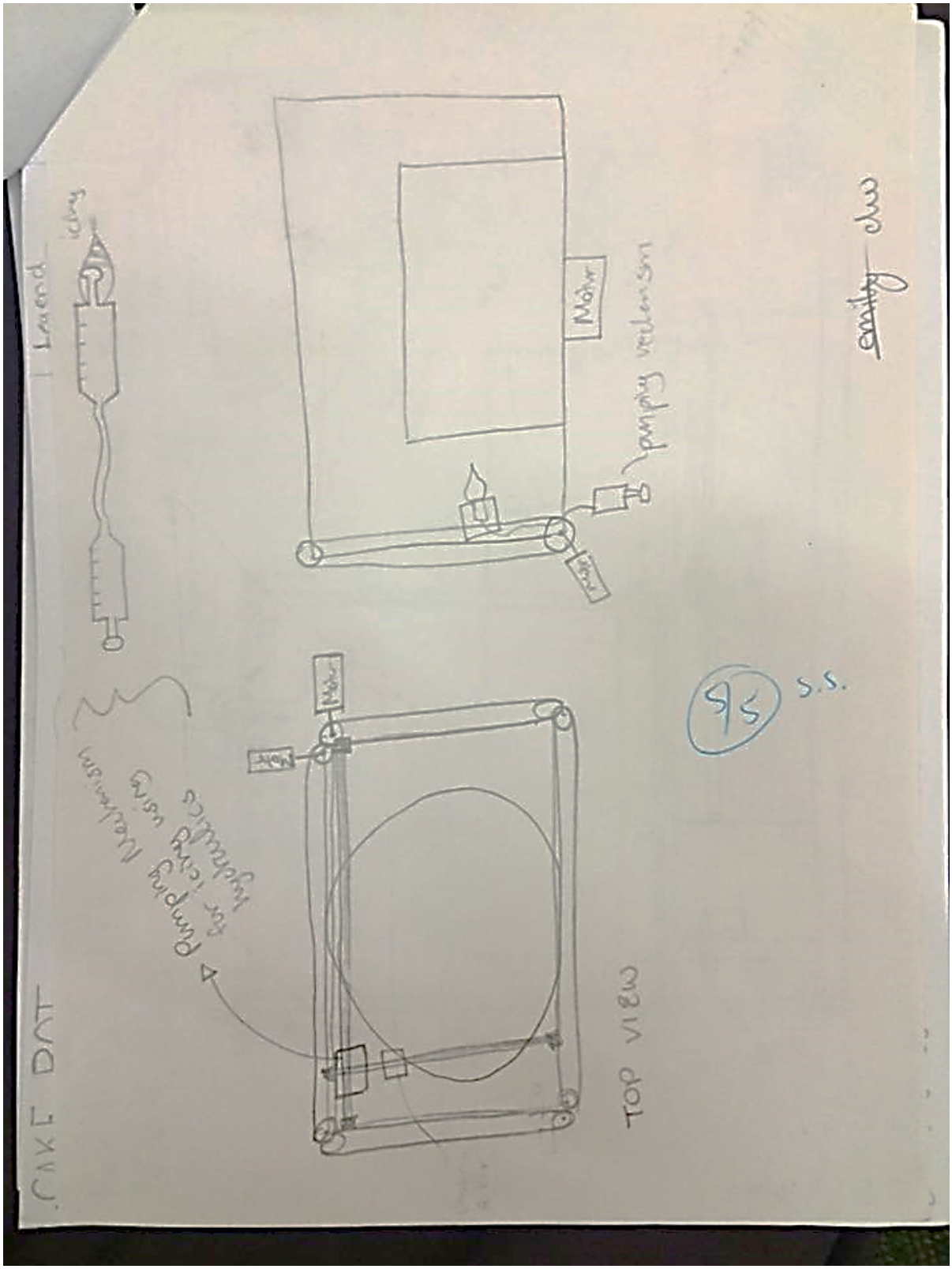


Figure: C.2: Cake Bot Concept 2

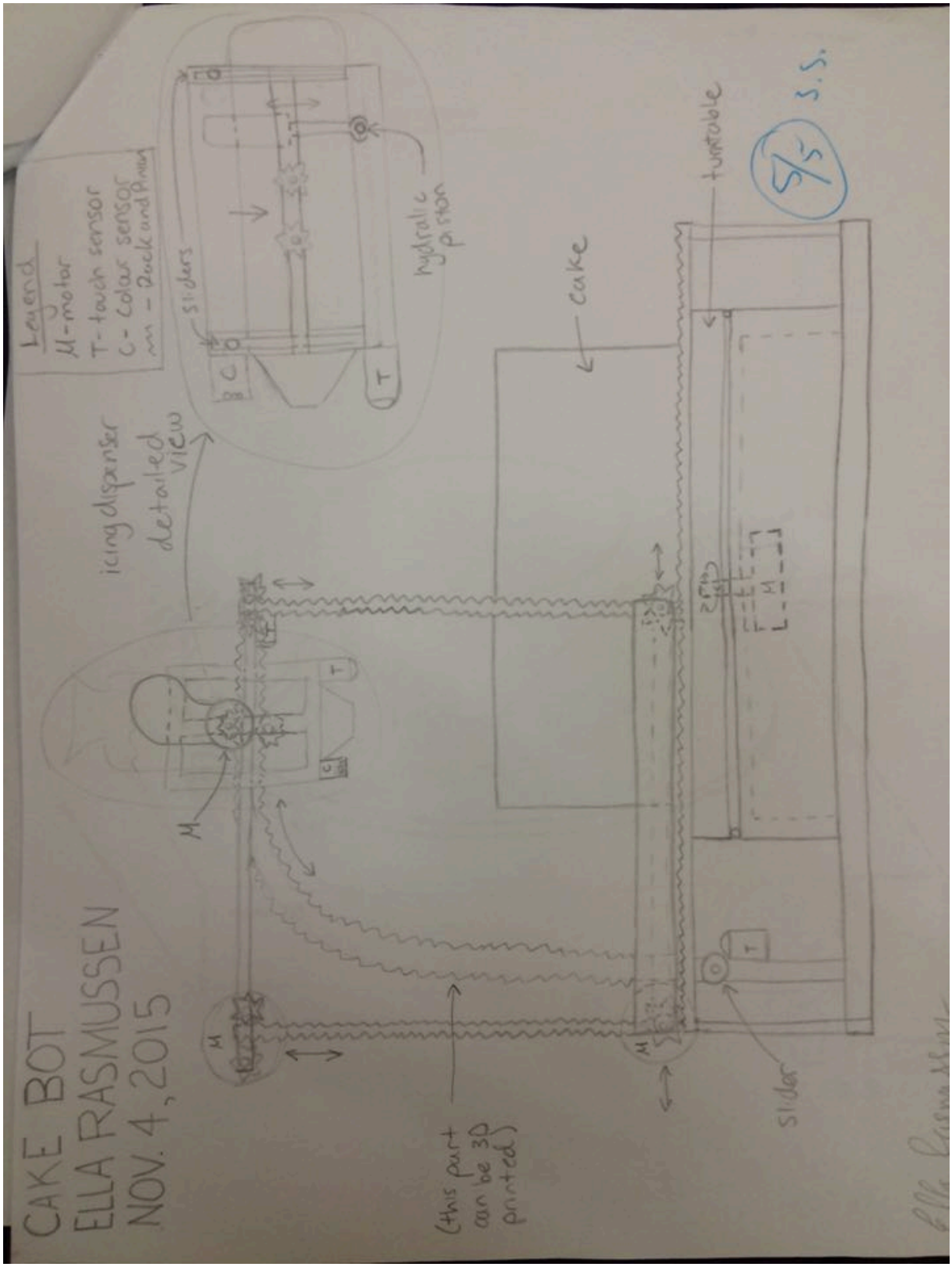


Figure: C.3: Cake Bot Concept 3

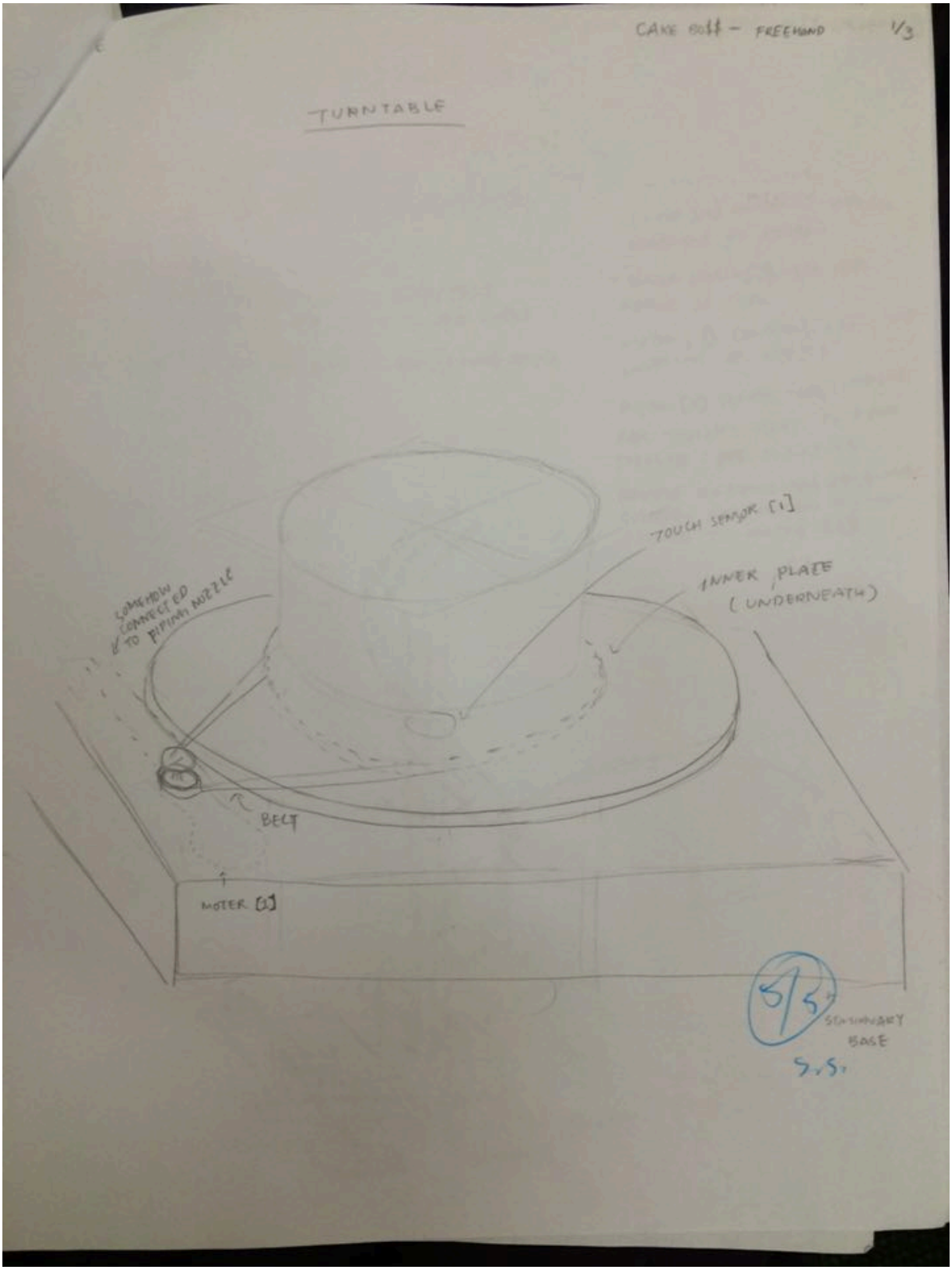


Figure: C.4: Cake Bot Concept 4a

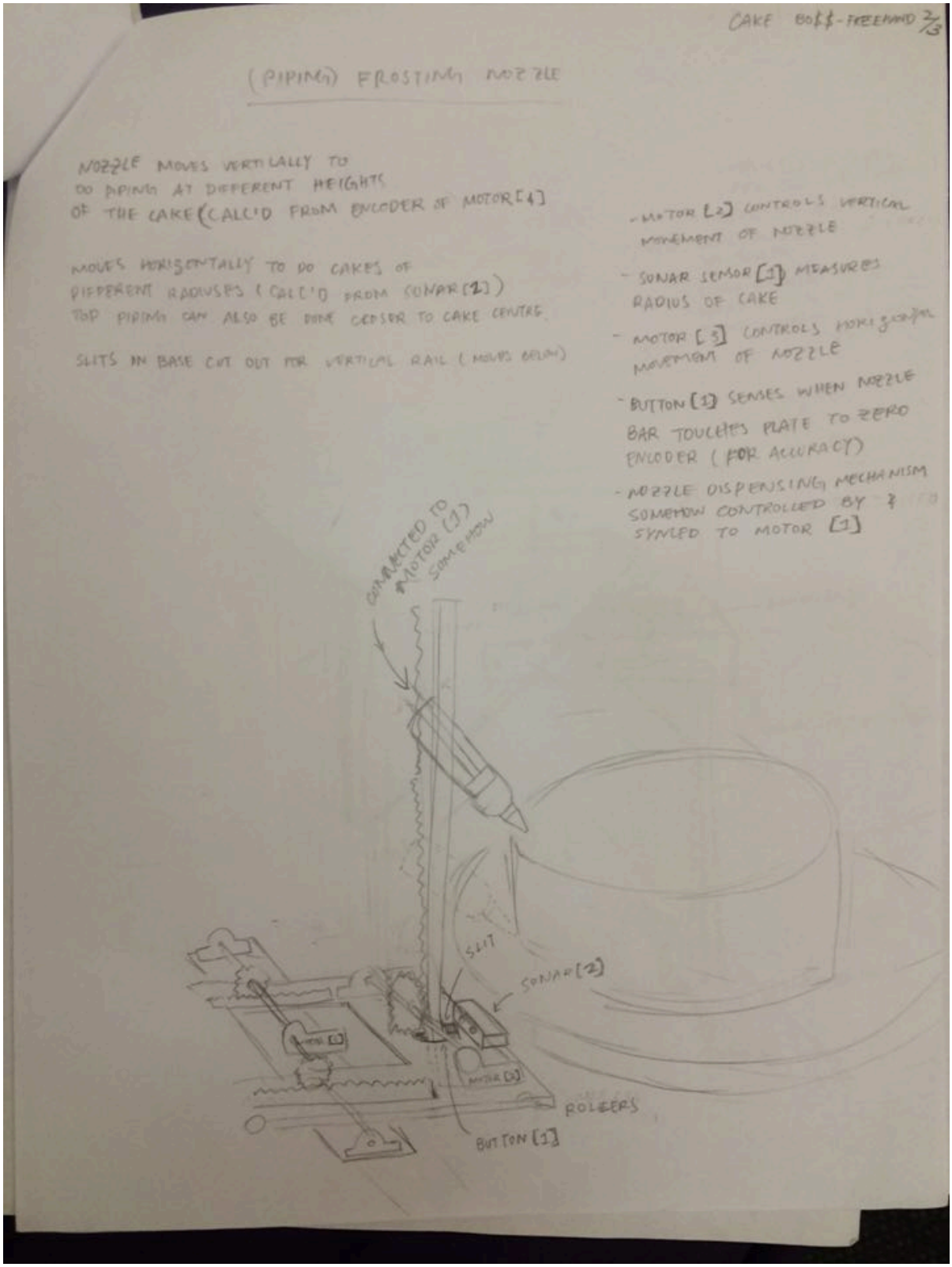


Figure: C.5: Cake Bot Concept 4b

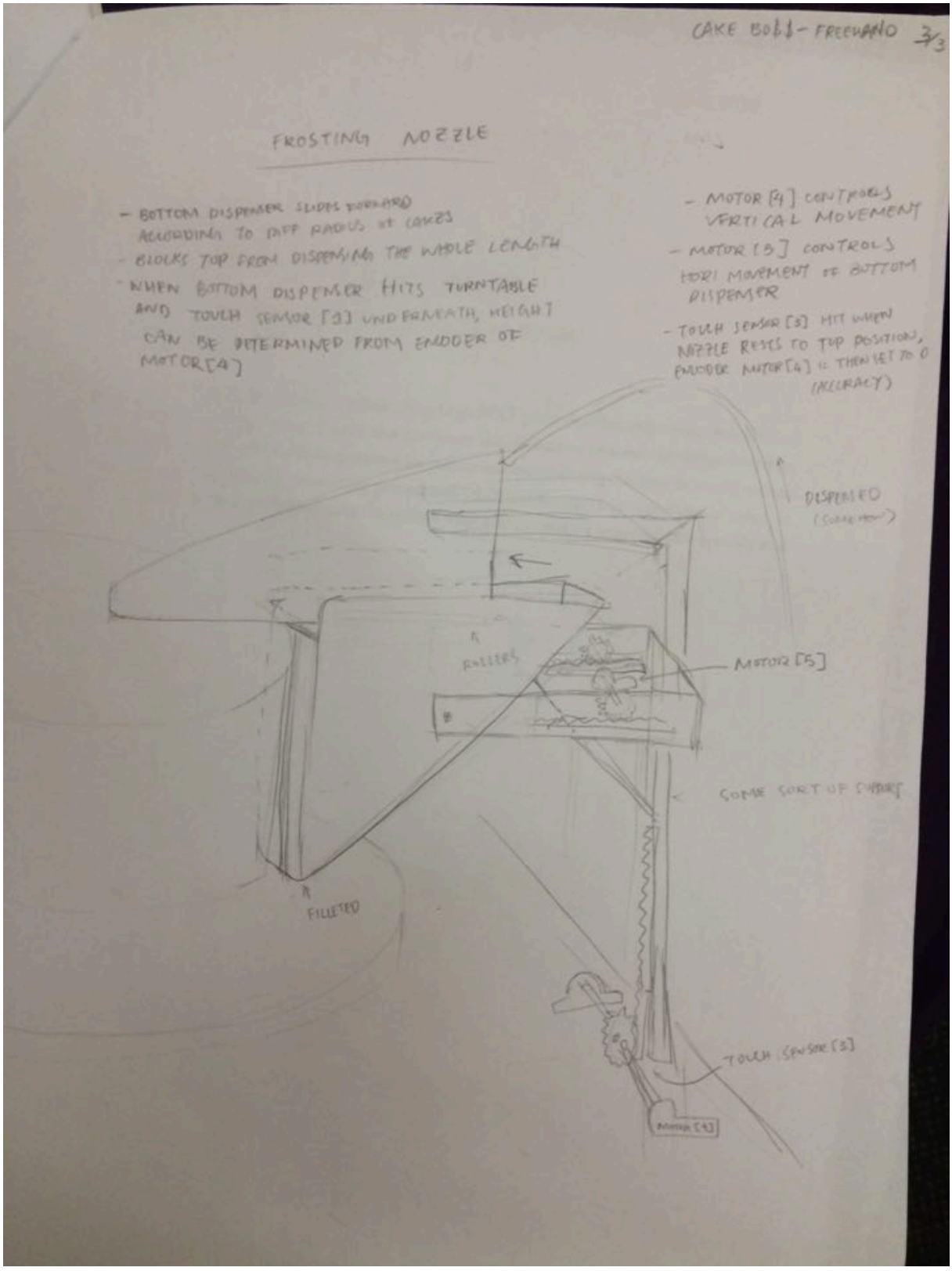
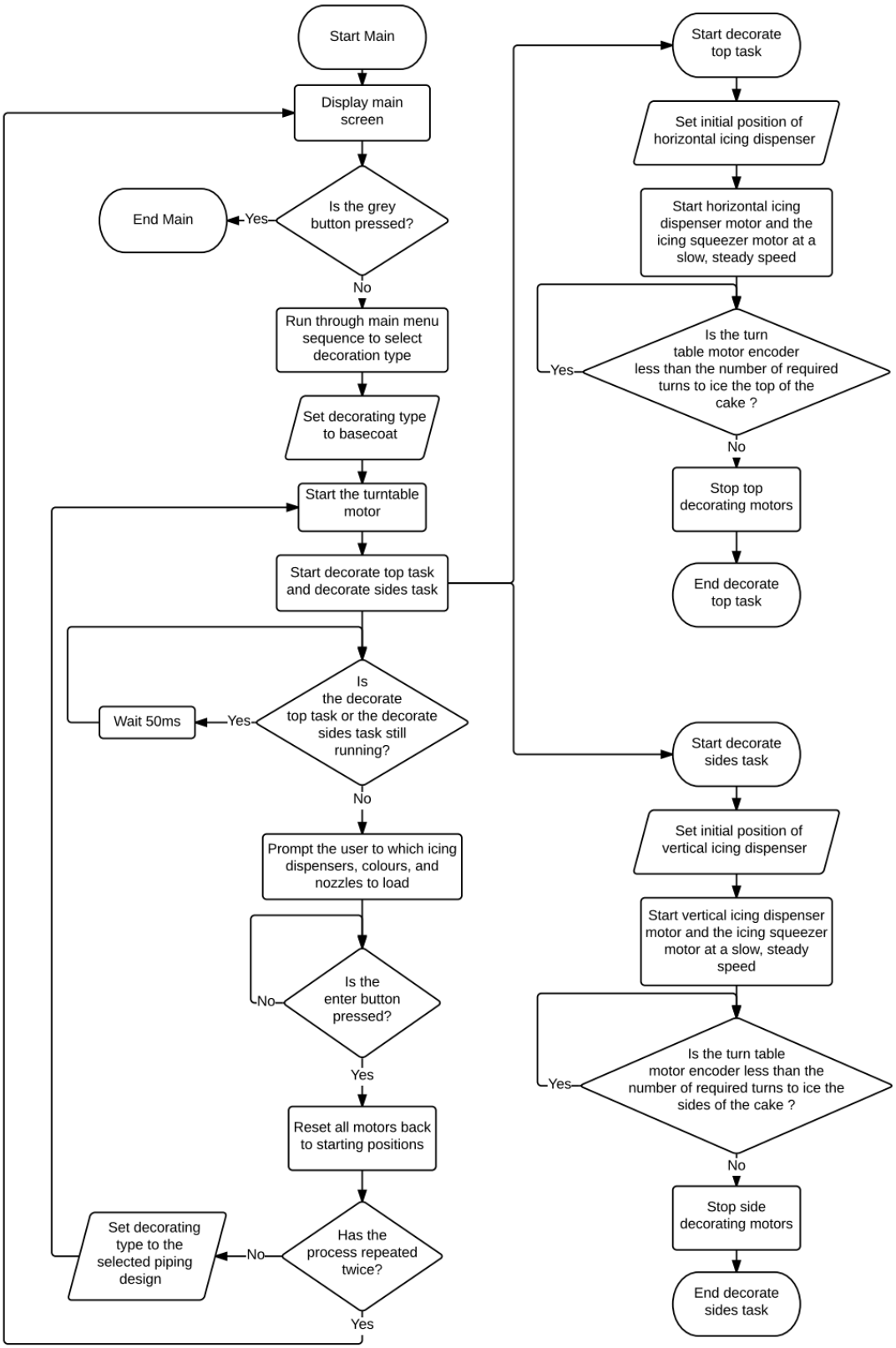


Figure: C.6: Cake Bot Concept 4c

Appendix D – Software Design Flowchart



Appendix E: Sample Calculations of Design Ratings

Design ratings for turntable mechanism:

Design A:

0.4*1 + 0.5*0.7 + 1*1 = 1.75

Design B:

0.4*0.5+0.7*1+1*0.2=1.1

Design C:

0.4*0.2+0.7*0.5+1*0.2=0.63

Design ratings for dispenser movement mechanism

Design A:

0.4*0.5+0.7*1+1*0.5=1.3

Design B:

0.4*1+0.7*0.2+1*0.2=0.74

Design ratings for dispensing icing mechanism

Design A:

0.4*1+0.7*0.5+1*0.2=1.75

Design B:

0.4*0.2+0.7*0.5+1*0.5=0.93