EV3 Robot Project

Final Report

Rosie, Victoria, Vivian and Jan

Dec 3, 2024

Table of Contents

List of Figures

List of Tables

Summary and Acknowledgements

Billie the Robot is designed to help elderly individuals manage their medication. It automates the process of dispensing pills at the correct times and dosages, making it easier for users to take their medicine. The robot uses sensors to navigate towards the user and dispenses pills in a more accessible way. Billie also provides reminders with sounds and displays important information about the medication on a screen. It ensures the user takes the medication by detecting when a pill is taken using a touch sensor. The software controls tasks like dispensing the pills, driving towards the user, and updating medication information. Overall, Billie aims to improve medication management, reduce errors, and support elderly independence.

Acknowledgements and thank you to Professor Hulls for creating the syntax sheet for file input/output in RobotC.

Section 1: Introduction

Older adults and the elderly often struggle with cognitive decline, which may lead to difficulties when taking medications themselves, such as taking correct medications on time and with the correct dosage [1].

In cases where a caregiver is giving medication to the elderly, they must learn about the medications that the individual is taking and the time of the medication distribution. This can be overwhelming for the caregiver. Billie the Robot enhances the medication intake process by ensuring the user is taking their medicine in accurate dosages.

Section 2: Scope

Section 2.1: Tasks

The following list describes all the tasks our robot completes:

- Read the 2 files with dosage information and medicine information
- Use timers to begin pill dispensing
- Display error messages when they occur (i.e. beacon not detected, not enough pills)
- Drive towards the user using the beacon and IR sensor
- Stop driving before reaching the beacon at a certain distance
- Dispense a certain number of pills specified by the dosage file
- Display medicine information on the screen (i.e. pill name, dosage, warnings)
- Detect touch sensor (user presses it to indicate pill is taken)
- Return to the original position
- Play sound
- Loop for several times to dispense all pills for the day then exit the program
- Write to the dosage file before shutting down

Section 2.2: Inputs

Billie the Robot uses file input to retrieve the user's dosage amount and information about the pill. It uses motor encoders and the gyro sensor to accurately travel towards the user. The position of the user is determined from the IR sensor and beacon. The user also presses the touch sensor to indicate that they have taken their pills.

Section 2.3: Interactions with the Environment

The robot uses 2 driving motors, 1 dispensing motor, EV3 speaker, and EV3 display to interact with the environment and people. The EV3 built-in speaker is used to play chimes for reminders to take pills or alerts for error. The EV3 screen displays medical information for the user.

Section 2.4: Shutdown Procedure

The robot reads the total amount of pills in the container, number of pills per dosage, and the frequency of each dosage from the input file. The robot automatically shuts down when all pills for the day have been dispensed.

Section 2.5: Changes in Scope

Originally, two beacons would be used to find the position of the user and return to the robot's original position. However, the IR sensor can only be connected to one beacon, so only one was used.

Additionally, an ultrasonic sensor was going to be used for object detection and for retrieving the distance from the robot to the user. Object detection was too complicated to implement in the time frame and the distance towards the user can be retrieved from the IR sensor and beacon.

The touch sensor was used instead of the built-in buttons on the EV3. This makes the button more accessible for elderly people.

File output was added later in the design process, since there needed to be a way to store the updated number of pills even after the robot shut down and rebooted.

On the mechanical side, using pulleys and levers to lift the tray higher for accessibility was changed to just a stationary tray since it was too elaborate to accomplish in the given time.

Finally, having the robot as two separate components, as show in Figure 1, was changed to just having one overall component. This decision was made since the sensors were inaccurate and it would be too difficult to accurately line up the moving component to the docking station.

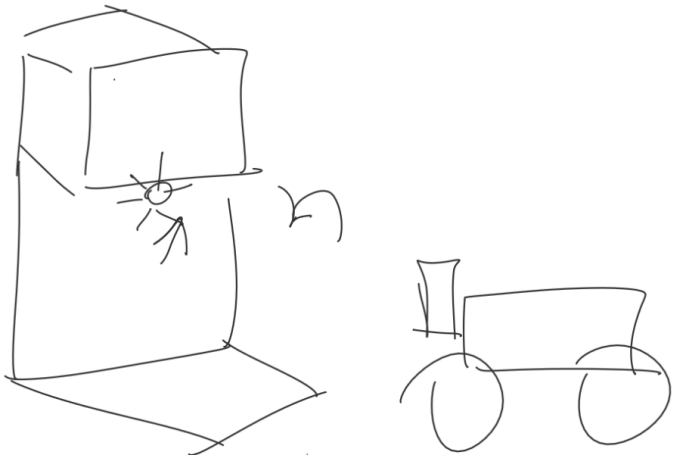

Figure 1 - Two Components Robot Idea

Section 3: Constraints and Criteria

Section 3.1: Constraints and Criteria

Constraints:

- Dispenses pills timely and in correct dose
- Drives accurately towards user (with beacon)
- Pills dispensed are accessible
- Make sure the pill is taken before leaving
- Display information for the medicine

Criteria:

- Robot needs to be balanced enough to easily drive around
- Limited resources with EV3: precision of sensors and motor movements
- Interactions (buttons and screen) need to be designed for elder's physical abilities

Section 3.2: Impacts

All constraints were impactful in guiding the design of the robot. They were all necessary components of the robot that needed to be met for it to fulfill its purpose properly and set up the base for the final designs.

Section 4: Mechanical Design and Implementation

Section 4.1: Overall Design

The robot is designed to easily drive around with all mechanical systems attached in a compact and balanced way. The main components of the robot are the EV3, sensors, driving motors and wheels, the dispensing system, pill container, ramp, and tray as shown in Figure 2 - Robot Overall Mechanical Design.

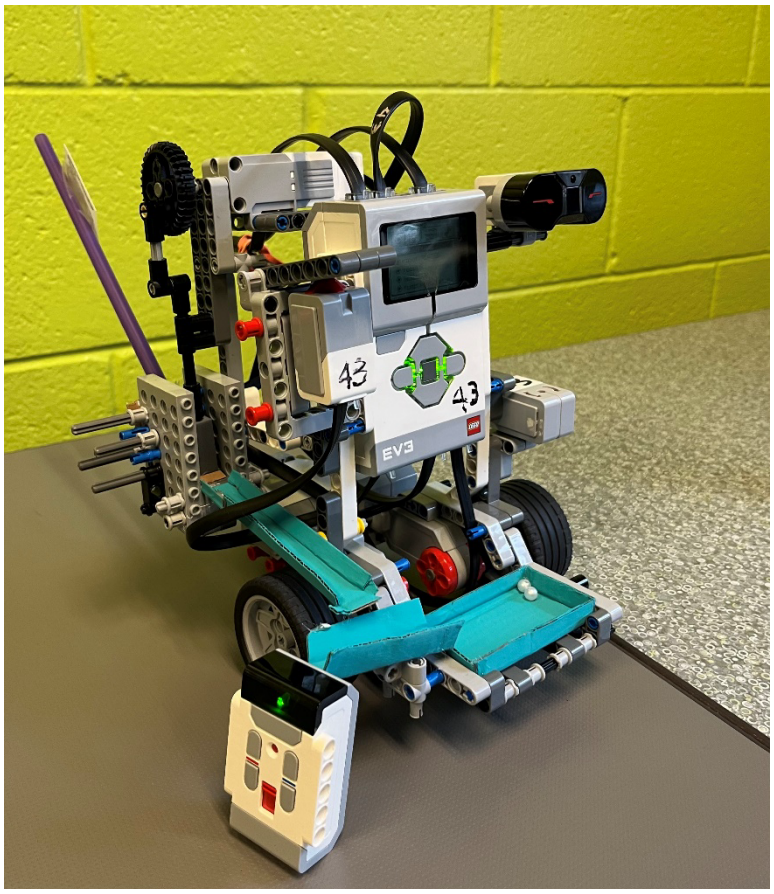

Figure 2 - Robot Overall Mechanical Design

The wheels and driving motors located on the left and right side of the robot enable it to drive and turn. The EV3 is positioned vertically to show the display. The tray is positioned at the front of robot, making the pills easy to access.

The sensors and dispensing system are positioned on the sides or behind the EV3 to avoid obstructing the view of the EV3 display and accessing the pills. The touch sensor is located

on the left side. The IR sensor and Gyro sensor are placed on the right side. The dispensing system has the motor and container behind EV3. To transfer the pill from the dispensing system to the tray, a ramp is built on the left side of EV3.

Section 4.2: Dispensing System

Section 4.2.1: Pill Container

The container was originally a vertical box, with an angled inside to collect the pills at the bottom, near the dispensing hole. The original box design is shown in Figure 3 - Vertical Pill Container.

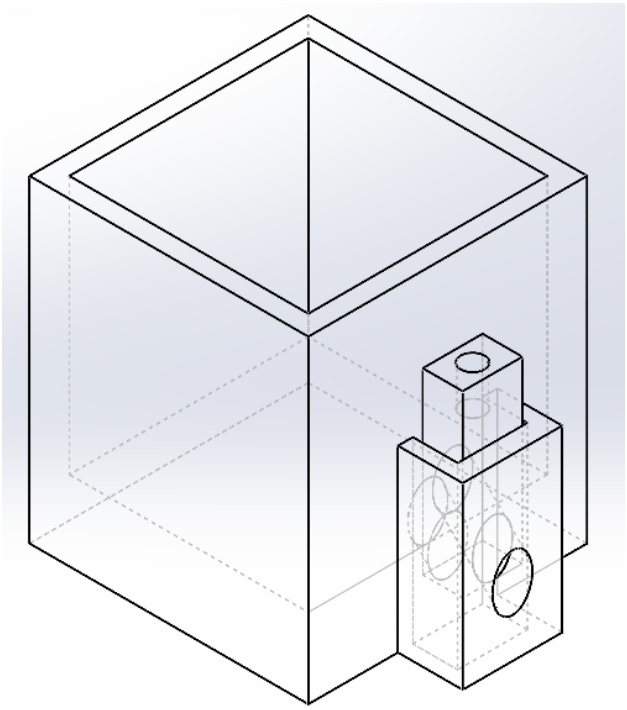

Figure 3 - Vertical Pill Container

However, this design had some problems. The pills, or the beads used for the demonstration and testing, would get stuck near the dispensing mechanism. This is because in this design, the beads don't line up in a single file, which causes crowdedness and spacing issues near the opening of the hole. In other words, multiple beads were blocking each other from being dispensed.

This problem was solved by a new, and final container design shown below.

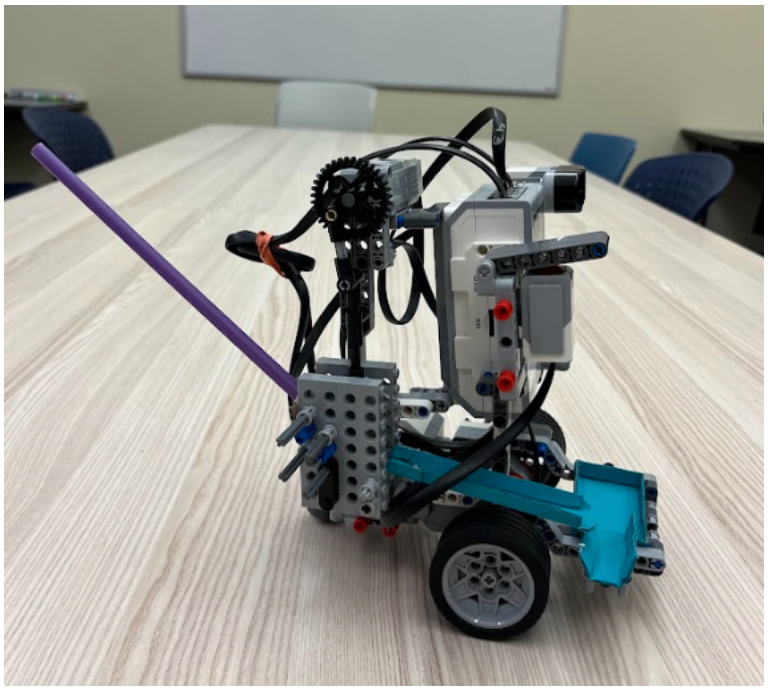

Figure 4 - Final Container Design

The pills are now being held in a straw, tightly held to the robot by a mechanism of squeezing plates and rods. The end of the straw is right above the ramp, so dispensed pills can effectively be dropped into the ramp and tray.

Section 4.2.2: Dispensing Mechanism

The robot features a slider-crank mechanism to dispense pills. A motor is used to rotate a gear with a rod connected to it. As the gear rotates, the rod moves linearly up and down or left and right. For reference, a diagram of the general slider crank mechanism can be found below.

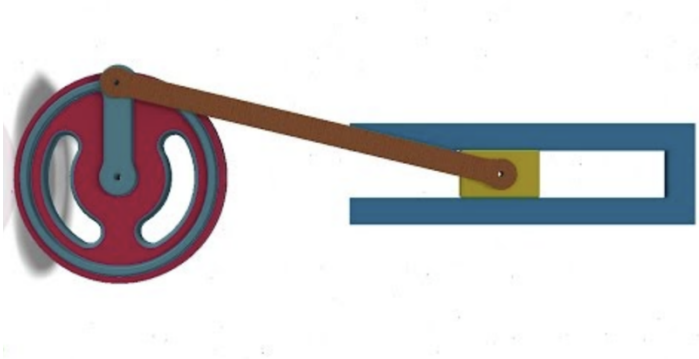

Figure 5 - Slider Crank Mechanism [2]

The mechanism attaches to a 3D printed component that blocks the opening of the pill container. So, when the rod moves up and down, the block moves out of the way, releasing a pill. A closer look at the mechanism can be found below.

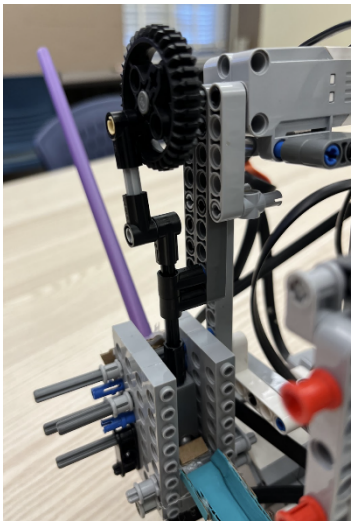

Figure 6 - Dispensing Mechanism

While testing, this system failed to dispense a single pill at a time, however, it was able to consistently dispense two at a time. So, the dosage info file was changed accordingly.

Section 4.3: Sensors

The project uses three different sensors, the Gyro, IR, and Touch Sensors.

The gyro is used to rotate the robot towards the specific angle of the beacon, which proved to be difficult in practice, because of the inconsistency and inaccuracy of the beacon and sensor. The gyro is located right above the left wheel, to accurately detect the rotation angle.

The IR Sensor is located above, and to the right, of the EV3 brick. This ensures that it can sense the beacon at a reasonable height. The IR sensor was also unable to switch between channels in the code (see Section 5.3.1 Rotating with IR Sensor), so the potitioning was modified.

The touch sensor was implemented to allow the user to press a button and tell the robot that the medication has been taken. The touch sensor is a button located at the side of the robot, modified with a panel on top, to increase the surface area, making it easier to press for the elderly.

Section 4.4: Ramp and Tray

A ramp and tray were designed to make the pills dispense at a more accessible position for the elderly to access. The ramp would connect directly to the opening of the pill container, and it would continue straight down until it reached the tray as shown below. The chassis was built to hold the tray securely with an upper lip right below the EV3 screen.

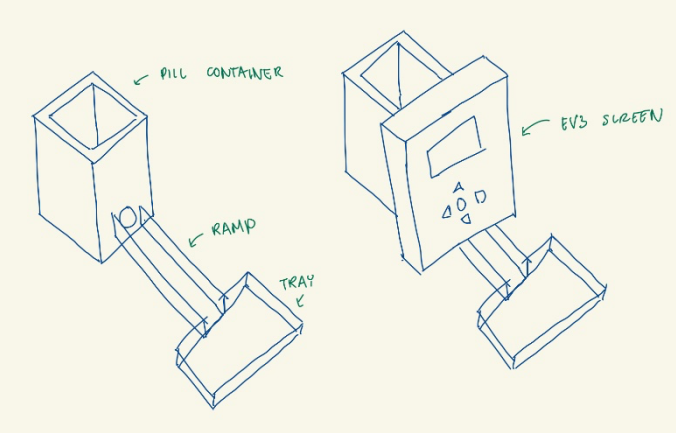

Figure 7 - Ramp and Tray Design

Earlier designs had the ramp in the centre of the robot since the container was originally sitting directly behind the EV3 screen. After the container was moved to sit on the left of the EV3 screen, the ramp was also modified to incorporate a turn as shown in Figure 8 - Final Ramp and Tray System.

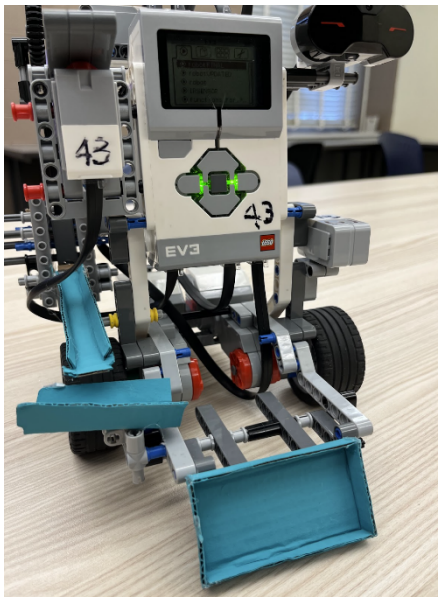

Figure 8 - Final Ramp and Tray System

The ramp and tray were supposed to be laser cut for precision and made from more durable materials. However, that would prove to be too time consuming, so to simplify the process, they were simply made of scrap cardboard and scissors.

While testing, the beads were difficult to grab as they would roll around in the tray. So, the tray was modified to be detachable, allowing for users to pour the pills into their hands.

Section 5: Software Design and Implementation

Section 5.1: Software Design

Section 5.1.1: Overall Program Breakdown

The software has 6 general tasks: starting up, error detection, driving towards the user, providing pills, driving back to original spot, and shutting down.

For starting up, the sensors are configured, files are read for medical information, and the program starting time is recorded as reference for dispensing pills later. During error detection, the pill information is verified, and the beacon is checked to be detectable. For driving towards the user, the program waits until the dosage time is reached and rotates the robot towards the user. The robot then drives straight before stopping a certain distance from the user. For providing pills, certain number of pills is dispensed, the medical information is shown, and a chime is played to remind the user to take the pill(s). The robot also detects if the touch sensor is pressed to determine if the user took the pill(s). For driving back, the robot turns 180°, drives the same distance back, and rotates into the original orientation. For shutting down, if the user has not taken all their pills for the day, an error message is displayed, and it plays a warning sound. The file is updated for the current number of pills, and a powering off sound is played.

Section 5.1.2: Tasks

- Reads the files with dosage and medical information
- Times when to drive and dispense pill
- Drives towards the beacon and stops at a certain distance from the beacon
- Dispense by moving motor
- Display information on EV3 screen
- Detect touch sensor
- Return to the original position

- Rotate to a certain angle
- Play chime/warning
- Display errors on EV3 screen
- Update the dosage information in file
- Loop until the dosages are all given for the day then exit the program

Changes made to the task list from earlier: changed from detecting EV3 button into detecting touch sensor (see Section 2.5: Changes in Scope), added the update file task to reflect the number of pills in the container, added the display error message feature, changed the shutdown procedure to happen after all dosages are dispensed for the day.

Section 5.1.3: Functions and Unit Testing

All the functions are listed with the return type, parameter(s), and author(s). Below each function are their description and unit testing for the function.

`void configure()` by Jan

Configure sensors including touch, gyro angle mode, IR sensor in beacon mode.

- Display values of sensors on EV3 screen after sensors are configured.

`void rotate(int angle)` by Vivian

Use the gyro sensor to rotate a certain angle.

- Make sure the robot rotates the same angle as the parameter.

void rotate() by Jan, Victoria, and Vivian

Use IR sensor and the beacon direction value to rotate towards the beacon.

- Display the IR sensor proximity and direction values on EV3 screen.
- Check if the robot is turning in the correct direction by moving the beacon around.
- Check if the robot stops turning when beacon is in front of the sensor.

`void drive(int stopProximity)` by Victoria

Start both motors to drive straight. Use the IR sensor and beacon proximity value to compare with the stopProximity, a positive int. When the proximity value is less than stopProximity, stop the motors.

- Display the proximity value from IR sensor while driving to ensure it is measured.
- Compare the stopProximity value and the value on the screen when robot stops.

`void dispense(int dosage)` by Vivian

Loop for the number of times indicated by the dosage parameter, to dispense pills. For each iteration, rotate the motor 360°, which dispenses exactly 2 pills.

- Set dosage to 1 and see if the motor turns 360°.
- Use pills to see if dispense(1) correctly dispenses 1 bead.
- Test with higher value to see if correct number of pills will be dispensed

`void returnRobot (int encoderCount, int turnAngle)` by Vivian

Wait a little, then drive the robot back to its original position and turn it back to face the original orientation. Start by rotating 180° degrees. Then drive straight for the encoder value indicated by the parameter, encoderCount. Lastly, use the turnAngle value, which is the angle the robot turned to face the beacon, to rotate to the original orientation.

- See if the robot can turn 180°.
- Mark the starting position of robot, then let it drive for a certain distance. Then see if the robot can drive back to the same location/orientation using the encoderCount parameter.
- Mark the starting orientation of robot, then let it turn for a certain angle. Then see if the robot can turn back to the same orientation using the turnAngle parameter.
- Combine the last 2 tests and see the overall performance of the function.

`bool readQuantity (int &totalNumPills, int &dosage, int &doseTime)` by Victoria, Rosie

Store int values from a file into variables. Variable, totalNumPills, store the number of pills in the container; dosage stores how many pills to dispense each time; doseTime stores how many seconds in between dispensing.

- Make sure the file can be opened and read by the program.
- Display the variables that the values are stored in after reading.

`void readMedInfo()` by Rosie

Store strings from a file into a string array.

- Make sure the file can be opened and read by the program.
- Display the string array that the information is stored in after reading.

`bool pillTaken()` by Rosie

After the pills are dispensed, wait several times for the button to be pushed, if not pushed, play an alarm and wait some more. While waiting, display the medical information. At the end, if the touch sensor is not pressed, return false, indicating the medication is not taken. Otherwise return true.

- See if the medical information is displayed on the screen while waiting for touch sensor to be pressed.
- Test if the alarm will be played while displaying each medical information.
- Display the Boolean value of the touch sensor to ensure it is measured correctly.
- Display the Boolean return by the function.

`void signalError(string errorMsg)` by Vivian

Signal there is an error in the dispensing process or the environment. Display the errorMsg on EV3 screen and play an alarm sound. Wait for a while so user can come to read the error message.

- Test if warning sound will be played.
- Pass all the error messages as parameters as shown in Table 1, making sure they are displayed correctly, and no text is cut off.

Table 1 - Error Message Test Cases for signalError

Error messages:
"CAN'T OPEN QTY"
"Fix QTY file!"

"ADD PILLS"
"IS BEACON ON"

`void updateFile(int totalNumPills, int dosage, int doseTime)` by Rosie

Before the program exits, update the information in the robot to reflect the pill state.

- Before changing the value in file, display the variables. After updating the file, read the files and display the values in the file to check if they are consistent.

Section 5.1.4: Data

Some global variables are used for motor driving and rotating power and file reading (see Appendices A: RobotC Source Code, line 6-10). In main, there are values storing dosage information, error messages, and program reference time (see Appendices A: RobotC Source Code, function: main).

Section 5.2 Testing

There is unit testing for each function and integration testing. See Section 5.1.3: Functions and Unit Testing for unit testing.

Integration testing of the robot is conducted with all the necessary text files stored on the robot, the pills placed in the container, and the sensors and motors properly connected. There are multiple test cases done as shown in Table 2. Each test case varies with the dosage information (see Section 5.1.3: Functions), whether the beacon is turned on, and if the touch sensor is pressed at different times of dispensing. The expected results are explained for each scenario. For dispense count, it specifies whether it's the first, second, or subsequent time the robot dispenses in that test case. For testing purposes, 120 seconds represents 1 day (so if a pill is taken every 40 seconds, the robot dispense 3 times).

Table 2 - Test Cases for Integration Testing

Test Case #	Dosage Info	Dispense Count	Beacon on/off	Touch sensor pressed	Expected Result

1	totalPills = 6 dosage = 2 doseTime = 40	1	on	Yes	Pills are dispensed and taken; robot returns after button is pressed
		2		No	Pills are dispensed and not taken; robot returns after it waits for a certain time
		3		Yes	Previously dispensed pills are still in tray, no pill is dispensed; robot returns after pills are taken and button is pressed
3	totalPills = 2 dosage = 2 doseTime = 40	1	off	N/A	The robot detects the beacon proximity value as 0 for multiple times, signaling the error "IS BEACON ON", waits for some time, then shuts down
2	totalPills = 2 dosage = 2 doseTime = 40	1	on	Yes	Pills are dispensed and taken; robot returns after button is pressed (no pills left after dispensing)
		2		N/A	The robot detects that there in so more pills in the container, signaling the error to "ADD PILLS", waits for some time, then shuts down

Section 5.3 Problems and Resolutions

Section 5.3.1 Rotating with IR Sensor

Since the IR sensor and beacon were not covered in the lectures, the functions available were tested and the initial idea was to use multiple channels to measure 2 beacons on 2 channels. However, during testing, it was discovered that the robot cannot measuring a specific channel using the function: `getIRBeaconChannelDirection(nDeviceIndex, nChannel)`. Only the function, `getIRBeaconDirection(nDeviceIndex)`, works. The working function measures the direction of channel 1 only. So, the decision is made to only use a beacon on channel 1 to drive towards the user.

So, the rotate function that measures the IR sensor uses a while loop to measure the direction value and pause the rotation when the value is 0. However, the robot would stop immediately when the function is called without turning until the expected orientation facing the beacon. The problem was identified through testing by displaying recent direction values. The measured value was not consistent and sometimes jumps to 0 due to limitations of the sensor. Through including both the proximity and direction values as conditions for the while loop, it does not stop for times when the beacon is not detected, which caused the stopping of rotation.

With the correct pausing condition for the IR sensor, the robot was going towards the correct direction and pauses when facing the beacon. However, it sometimes turns back and forth a lot of times before pausing. So, the range of acceptable directions to pause is expanded from only 0 to any value from -1 to 1. After that change, the robot could rotate correctly and smoothly towards the beacon.

Section 5.3.2 Storing to String Array in Function

The function `readMedInfo()` reads strings from a text file and stores them in a string array. Initially, passing the array by reference was attempted but did not work. Assistance was requested from Ryan, who explained that using a global string array would be a suitable solution. So, a global string array was created, and the function now directly stores the file's values in this array.

Section 6: Verification

Section 6.1: Constraints Met

Below are the constraints set for the project demo and how they were met.

1. Dispense pills timely

   A timer was used in the program that counted the time between pill dispenses to ensure they were dispensed at the correct times.

2. Ensure pill is taken before leaving

   The button can be pressed after the sound plays when the pill is dispensed. When the button is pressed, it means that the pill has been taken, and the robot can operate the next dispense as usual. If the button is not pressed, this will prevent pills from being dispensed the next time, preventing overdose.

3. Display information for the medicine

   Information from the file is displayed in a large font when pills are dispensed. The screen where the medical information is displayed is facing the same direction as the pill tray, making them both accessible from the same position.

Section 6.2: Constraints Not Met

1. Drive accurately towards user

   Due to limitations with the EV3 technology, driving toward the user was not extremely accurate. Sometimes, it would not stop right in front of the user but be a couple of centimeters off to one side. A fix to this issue is to hold the beacon level with the IR sensor, which improves the accuracy.

2. Pills dispensed are accessible

   To access the pills, the user needs to bend down and pick them up from the ground level, which is not very accessible. Originally the design had planned to include a mechanism that would raise the pills up to the user, however that was not executed

because of technical limitations. Instead, the final design included a removable pill tray, which makes the pills slightly more accessible.

3. Dispense pills in correct dose

   In the final robot, the dosage was limited to multiples of 2, since that was the amount dispensed in one revolution of the gear on the extending arm. The constraint failed to be met due to limited testing of the dispensing mechanism in the early stages to ensure it dispensed one at a time.

Section 7: Project Plan

Section 7.1: Task Distribution

All the team members worked on both mechanical and software aspects of the project.

For the mechanical design, see Table 3 for the task distribution of the 5 main components.

Table 3 - Mechanical Design Task Distribution

Component	Chassis & EV3	Dispensing System	Pill Container	Sensors	Ramp and Tray
Team Member	Victoria Vivian	Vivian	Jan Vivian Rosie Victoria	Vivian	Rosie Victoria

For software, the functions are split between team members to work on (see Section 5.1.3: Functions and Unit Testing).

Section 7.2: Revisions to the Project Plan

The original project plan was developed so at least a week would be dedicated hardware, software, and other presentation requirements. The original project plan is defined in the table below.

Table 4 - Original Project Schedule

Oct 29 - Nov 2:	• Hardware (Tuesday during the rest of tut time, Wed & Sat meeting)
Nov 4-Nov 9: Nov 5: Physical system formal presentation	• Hardware (Wed meeting) • Software rough structure (Sat meeting)
Nov 11-Nov 16: Nov 13/15: Software Presentation	• Software most of functions done (Wed & Sat meeting)

Nov 18-Nov 23: Nov 19: Final demonstration	• Wrap up hardware and software (Wed & Sat meeting)
Nov 25-Nov 30:	• Report (Wed & Sat meeting)
Dec 2-Dec 3: Dec 3: Final Report Due	• Finish reports (anytime)

Section 7.3: Project Plan vs Execution

The software presentation was moved a week before the anticipated date. This caused some delays in the schedule as hardware was postponed until after the software presentation and most of the code was finalized. Other delays were experienced with the original plans for the hardware. Redesigning the container, along with delays in the ramp and tray, compounded by long wait times and printer unavailability for laser cutting, ultimately forced a switch to cardboard (see Section 4.4: Ramp and Tray), further delaying the schedule.

The schedule unfortunately did not account for the redesign and delays, though a few work sessions and meetings led to a ready and finished product for the final demonstration.

Section 8: Conclusions

The pill dispensing robot helps elderly take their pills in a timely manner by dispensing them in the correct dosage at the proper times. The design was able to meet most constraints and criteria to achieve this purpose, although with limitations, both due to technology and overlooked design aspects.

Main components of the mechanical design include a body that drives, on which contains storage for pills waiting to be dispensed, a dispensing mechanism, a ramp and tray, a button for click when pills are taken, and a screen to display medical information. There were some changes from the original design, especially the dispensing storage, which changed from a 3D printed box to a tube that could hold the pills in a single file line.

Main features of the software design include a file that is used to determine the dosage amount and times, and the information displayed on the screen. There is also the driving aspect, with programs that enable the robot to drive towards the user, stop a certain distance away, and retrace its steps back to the starting point when it is done dispensing. There are also features that play alerts when pill storage is low and procedures for when the pills are not taken. Then when all the pills for the day have been dispensed, the robot returns to the starting position and shuts down.

Section 9: Recommendations

Section 9.1: Mechanical Design Improvements

The EV3 screen was made to be perfectly vertical so that it's more visible. While testing, it was hard to read the screen since the user would have to bend down to be eye level with the robot to read the display. For next time, the screen can be slightly tilted so it's easier to read at different angles, as seen below.

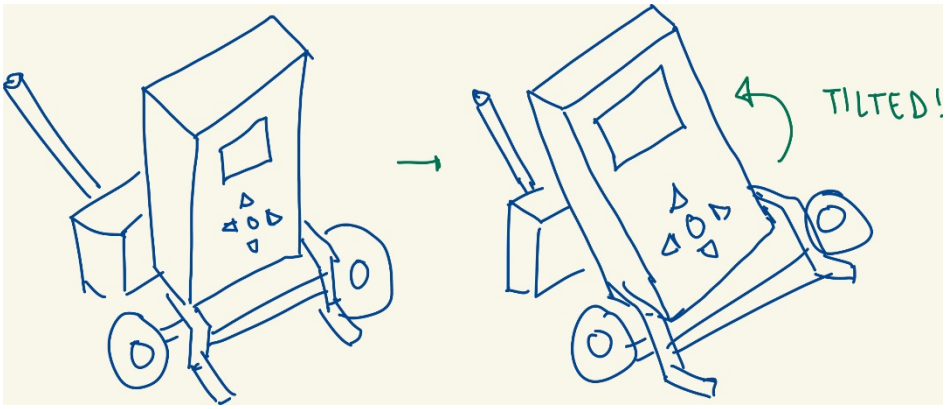

Figure 9 - Tilted EV3 Screen

More supports can also be added for the dispensing mechanism as the robot currently shakes when the pills are dispensed. This will make the overall design more robust and sturdier.

In the future, the container for the pills will have to be redesigned. As of now, the pills are being stored in a plastic straw, which is not the most durable option. The length of the straw would also have to increase as the number of pills increase, reducing scalability. A solution to this is shown in Figure 10 - Container Concept, as having a stack of these plastic tubes could increase the number of pills being stored.

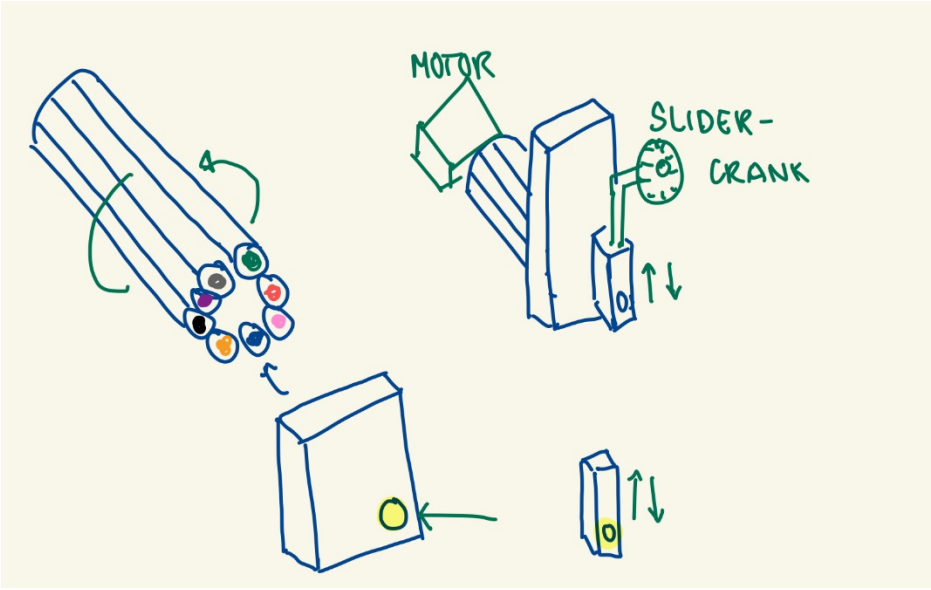

Figure 10 - Container Concept

Essentially, pills will be stored single-file in each cylindrical tube and the stack of tubes can rotate as each tube empties. This concept also accommodates multiple different types of medications. Currently, the robot is only able to store one type of medication, but in practice, people often have multiple prescriptions. So, depending on the dose, the robot can rotate the cylindrical shaft and dispense the correct medication. A funnel can also be made for the top of the container, to make refilling pills easier.

In the future, the robot could also raise itself, or raise the dispensed pills at a higher height, so patients don't have to bend down. This feature could make the pills and robot even more accessible to the patients.

Section 9.2: Software Improvements

A software improvement could be allowing interactions with the EV3 buttons for adding pills or changing the dosage information. Currently, the only way to modify the number of pills is using a laptop to upload files to the robot, which is very inconvenient. Changing these values through buttons on the robot makes it easier to update the quantity for users or their care givers.

References

[1] C. Salzman, ”Medication compliance in the elderly,” PubMed, https://pubmed.ncbi.nlm.nih.gov/7836347/ (accessed Dec. 3, 2024).

[2] Skyline Tutorials, “How Slider Crank Mechanism Works,” YouTube, https://www.youtube.com/watch?v=ZO8QEG4x0wY&ab_channel=SkylineTutorials (accessed Nov. 4, 2024).

Appendices A: RobotC Source Code

```
#pragma SystemFile
#define _EV3FILEIO 1
#include "PC_FileIO.c"

//global vars
const int DRIVE_POWER = 30;
const int ROTATE_POWER = 10;

const int F_LINES = 4;
string medInfoList[F_LINES]; //Ryan told us to use global string array

void configure()
{
	// gryo
	SensorType[S3] = sensorEV3_Gyro;
	wait1Msec(50) ;
	SensorMode[S3] = modeEV3Gyro_Calibration;
	wait1Msec(50);
	SensorMode[S3] = modeEV3Gyro_RateAndAngle;
	wait1Msec(50);
	// touch
	SensorType[S1] = sensorEV3_Touch;
	wait1Msec(50);
	// IR sensor
	SensorType[S4] = sensorEV3_IRSensor; // configure type & MODE
	wait1Msec(50);
	SensorMode[S4] = modeEV3IR_Calibration;
	wait1Msec(50);
	SensorMode[S4] = modeEV3IR_Seeker;
	wait1Msec(50);
}

void rotate(int angle) {
	resetGyro(S3);
	if (angle < 0)
	{
		motor[motorA] = ROTATE_POWER;
		motor[motorD] = -ROTATE_POWER;
	}
	else
	{
		motor[motorA] = -ROTATE_POWER;
		motor[motorD] = ROTATE_POWER;
```

```
	}

	while (abs(getGyroDegrees(S3)) < abs(angle)) {}

	motor[motorA] = motor[motorD] = 0;
}

void rotate()
{
	eraseDisplay();
	displayCenteredBigTextLine(5, "Locate Beacon");

	int direction = 0;
	while (direction == 0 || direction == 100) {
		direction = getIRBeaconDirection(S4);
	}
	int distance = 0;

	wait1Msec(1000);

	do
	{
		direction = getIRBeaconDirection(S4);
		distance = getIRBeaconStrength(S4);

		wait1Msec(1);
		if (direction > 0)
		{
			motor[motorA] = -ROTATE_POWER;
			motor[motorD] = ROTATE_POWER;
		}
		else if (direction < 0)
		{
			motor[motorA] = ROTATE_POWER;
			motor[motorD] = -ROTATE_POWER;
		}

		eraseDisplay();
	} while (!(direction < 2 && direction > -2 && distance != 0
			&& distance != 100));

	motor[motorA] = motor[motorD] = 0;
	wait1Msec(1000);
}

void drive(int stopProximity)
```

```
{
	nMotorEncoder[motorA] = 0;
	motor[motorA] = motor[motorD] = DRIVE_POWER;
	while(getIRBeaconStrength(S4) > stopProximity) {}
	motor[motorA] = motor[motorD] = 0;
}

void dispense(int dosage)
{
	for (int i = 0; i < dosage; i++)
	{
		nMotorEncoder[motorB] = 0;

		motor[motorB] = 10;
		while (abs(nMotorEncoder[motorB]) < 360) {}
	}

	motor[motorB] = 0;
	wait1Msec(25);
	// avoid turning off motor did not work, wait and try again
	motor[motorB] = 0;
}

void returnRobot(int encoderCount, int turnAngle)
{
	wait1Msec(1000);

	rotate(180);
	wait1Msec(500);

	nMotorEncoder[motorA] = 0;
	motor[motorA] = motor[motorD] = DRIVE_POWER;

	while (abs(nMotorEncoder[motorA]) < encoderCount) {}

	motor[motorA] = motor[motorD] = 0;

	if (turnAngle < 0)
	{
		rotate(-(180+turnAngle));
	}
	else
	{
		rotate((180-turnAngle));
	}
}
```

```
bool readQuantity(int &totalNumPills, int &dosage, int &doseTime)
{

	TFileHandle fin;
	bool fileOkay = openReadPC(fin, "dosageInfo.txt");

	if (fileOkay)
	{
		readIntPC(fin, totalNumPills);
		wait1Msec(100);
		readIntPC(fin, dosage);
		wait1Msec(100);
		readIntPC(fin, doseTime);
		wait1Msec(100);
	}
	else
	{
		return false;
	}
	closeFilePC(fin);
	return true;
}

void readMedInfo()
{
	TFileHandle medFile;
	bool fileOkay = openReadPC(medFile, "medInfo.txt");

	//read from file
	if (fileOkay)
	{
		for (int i = 0; i < F_LINES; i++)
		{
			readTextPC(medFile, medInfoList[i]);
		}
	}
	else
	{
		displayTextLine(0, "could not open medFile");
		wait1Msec(2000);
	}
	//close file
	closeFilePC(medFile);
}
```

```
bool pillTaken()
{
	int numLoops = 2;
	for (int i = 0; i < numLoops; i++)
	{
		for (int lineNum = 0; lineNum < F_LINES; lineNum++)
		{
			playSound(soundFastUpwardTones);
			displayCenteredBigTextLine(2, medInfoList[lineNum]);
			clearTimer(T1);
			while(time1[T1] < 2000)
			{
				//checks if pill taken button is pressed
				if (SensorValue[S1] == 1)
				{
					eraseDisplay();
					displayCenteredBigTextLine(1,
								"BUTTON PRESSED");
					wait1Msec(2000);
					//pill taken
					return true;
				}
			}
		}
	}
	//pill not taken
	return false;
}

void signalError(string errorMsg)
{
	eraseDisplay();
	displayCenteredBigTextLine(4, "%s", errorMsg);
	playSound(soundException);
	wait1Msec(10000);
}

void updateFile(int totalNumPills, int dosage, int doseTime)
{
	TFileHandle fout;
	bool fileOkay = openWritePC(fout, "dosageInfo.txt");

	if (fileOkay)
	{
		writeLongPC(fout, totalNumPills);
		writeEndlPC(fout);
```

```
            writeLongPC(fout, dosage);
            writeEndlPC(fout);
            writeLongPC(fout, doseTime);
      } else
      {
            displayBigTextLine(1, "bad dose file");
      }
      closeFilePC(fout);
}

task main()
{
      setSoundVolume(50);
      int totalPills = 0; //total pills in container
      int dosageFreq = 0; // pills dispensed per day
      int dosageNum = 0; //how many pills dispensed at each increment
      int doseTime = 0; // amount of time (in seconds) per dosage
      bool pillInTray = false; // whether pill is left in tray
      bool error = false; // if there is error
      string errorMsg = "";

      int startTime = nPgmTime;

      readMedInfo();
      error = !readQuantity(totalPills, dosageNum, doseTime);
      if (error)
      {
            errorMsg = "CAN'T OPEN QTY";
      }
      else if (dosageNum <= 0 || doseTime <= 0)
      {
            errorMsg = "Fix QTY file!";
            error = true;
      }
      else
      {
            dosageFreq = 120/doseTime; //store 120 in const
      }

      configure();
      resetGyro(S3);
      int leaveTime = 10;

      // repeat for the number of dosages of pill unless error occurs
      for (int i = 0; i < dosageFreq && !error; i++) {
            resetGyro(S3);
```

```
eraseDisplay();

// show number of pills left
displayCenteredBigTextLine(7, "# Pill: %d", totalPills);

// wait and show time from beginning of the program
while ((nPgmTime-startTime)/1000 < leaveTime)
{
     displayCenteredBigTextLine(5, "Time: %d",
                                   (nPgmTime-startTime)/1000);
}

// make sure beacon is turned on
// checking multiple times in case of bad sensitivity
int proximity = getIRBeaconStrength(S4);
for (int j = 0; j < 5 && proximity == 0; j++)
{
     proximity = getIRBeaconStrength(S4);
     wait1Msec(100);
}

// check for errors before driving
if (totalPills < dosageNum || proximity == 0)
{
     error = true;
     if (totalPills < dosageNum)
     {
          errorMsg = "ADD PILLS";
     }
     else
     {
          errorMsg = "IS BEACON ON";
     }
}
else
{
     rotate();
     int gyroAngle = getGyroDegrees(S3);
     drive(10);
     int travelEncoderCount = abs(nMotorEncoder[motorA]);

     if (!pillInTray)
     {
          dispense(dosageNum);
          totalPills -= dosageNum*2;
     }
```

```
            pillInTray = !pillTaken();
            eraseDisplay();
            returnRobot(travelEncoderCount, gyroAngle);

            leaveTime += doseTime;
      }
}

if (error) {
      signalError(errorMsg);
}

updateFile(totalPills, dosageNum, doseTime);
playSound(soundDownwardTones);

// wait until the powering off sound finish playing
while(bSoundActive) {}
```