ME 588 MECHATRONICS

Single-Plant Optimized Robotic Pollinator

Team 6

Jon Branch

Abhishek Nair

Aishvarya Joshi

Jack Ferlazzo

Meredith Bloss

Seth Honnigford

29th April, 2024

Abstract

This report discusses using finite state machine programming and sensory interactions to create an autonomous colored "pollen" sorting robot following the project guidelines. In this design, the robot sorted pollen using a color sensor and servo sorting system. Only one-color pollen, the pollen delivered to Plant 3, is accepted and delivered, with the rest rejected. Movement is achieved using a four-wheel system with two back wheels powered by 12V DC motors. The team designed and printed the robot chassis using SolidWorks computer-aided design, CAD. This design successfully met project requirements, delivering pollen autonomously and shutting down after 2 minutes of run time, while displaying information to the observing audience.

1. Introduction

The overall objective of this project was to build a robot which would move autonomously to locate Plants and deliver correct colored Pollen to them, as assigned by the TA. To accomplish this, there were multiple sub-systems designed by the team, each with different objectives. The power system consists of a battery and circuit integration which gives 12V to motors for the drive and 5V to H-Bridge and Raspberry Pi. The drivetrain where the motors and encoders are combined with the H-bridge which controls the direction of wheel rotation and the distance measured using the encoders. A sorting system using a color sensor used to detect pollen color, a storage system for the accepted pollen, and a system of servos that actuate to control pollen movement within the system. The display system uses LEDs to provide visual indicators indicating if the robot is in ON or OFF status. These LEDs also indicate the color of different plants which can be assigned to the robot using rotary encoders when the game starts. Finally, the software system uses an FSM and microcontrollers to implement logic for each sub system..

The strategy to accomplish the main objective is designed after simplification. The goal is redefined to only consider the color given to Plant 3 (the closest plant in terms of distance from the Barn). The Pollen color assigned to Plant 3 is accepted into the storage area and the rest of the Pollen balls are rejected outside in the Barn area. Upon starting the game, the robot travels forward and deposits the complete storage of pollen balls in the top frame of Plant 3. The robot then travels straight backward and enters the barn for refill. There is an emergency STOP button given which cuts off the power.

The robot body is 3D printed from PLA. The robot has three frames to it.The bottom most frame consists of 2 motor-encoder sets which drive the back wheels. The two front wheels are supporting wheels which rotate freely. The wires from the motor and encoder go up to the second frame. The middle frame is the main body of our robot. It consists of the power supply, Li-Ion Battery for motors, and Portable Charger Power Bank for 5V DC. It also has the placement of all circuits, namely, power circuit which converts the Li-Ion 14.8V to 12V output for the motors, H Bridge for the control of motors, Raspberry Pi for overall control, Arduino/QTPY board for color sorting mechanism control, bread board for the rotary encoders and display system. It also facilitates the emergency STOP button and the Game Start button. The top frame of the robot has the Pollen sorting system and the various displays as mentioned above. The human-robot interface lies in this frame where the rotary encoders are placed to manually enter the color assigned to each plant and input the different colored Pollen into the robot for sorting when it is in the barn. The distance from the barn to Plant 3 is 2 feet. Hence, the encoder pitch is calculated based on the rotation of the wheel. The count is calculated for 2 feet based on the circumference of the wheel and calibrated to get an approximate number after which the robot will stop once reaching the plant. The color sensor is calibrated using lux values tested for each color. The rotary encoder is used as an actuator to set plant input colors as it is rotated.

2. Detailed Robot Design

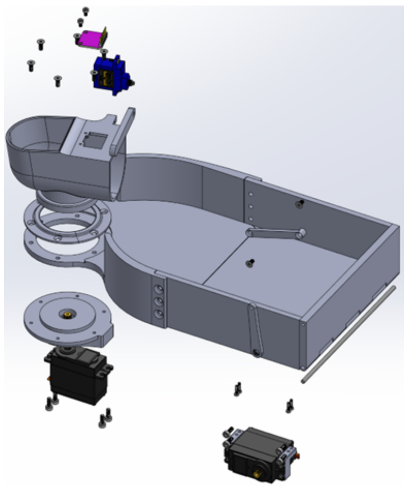
2.1. Game Strategy

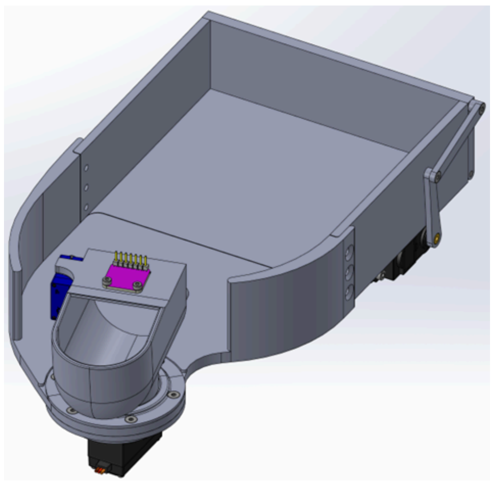
After hearing about the new timing rules where sorting time would not count toward the overall time, our strategy changed a lot. Since time only counted while the robot was moving or outside the starting area, we no longer cared about the time it took to fully sort pollen. The robot was also limited in size, so we wanted to maximize the amount of pollen it could store while still being able to deposit the pollen in the higher scoring area. With these two design considerations, we decided the best way to do this was to only store pollen designated for plant three. This way we wouldn't have to waste space for a sorting mechanism or three different storage chambers. Plant three is the easiest to get to from the starting area, meaning our robot only had to move forward or backwards. Theoretically, this would improve our ratio of pollen deposited per time spent, giving us a higher score than our competitors.

2.2. Physical Design

The sorting mechanism, seen in Figures 1 and 2, consisted of two servos, an SG90S micro servo, and a Miuzei MG996R Servo, acting as the gate and sorting servos, respectively, which worked with a TCS34725 color sensor placed above the resting point of the inserted ball. Since the strategy of this robot was to only store the ball color corresponding to plant 3, the sorting servo only had two positions, only activating if the inserted ball did not match the plant 3 color. If the inserted ball did not match the selected storage color, the sorting servo would rotate 70º to the right, where the gate servo would then rotate upwards from 0º to 90º, dumping the undesired ball. If the color did match the desired storage color, the sorting servo would not activate and the gate servo would open, allowing the ball to roll into storage.

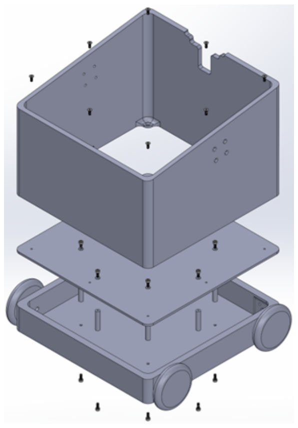
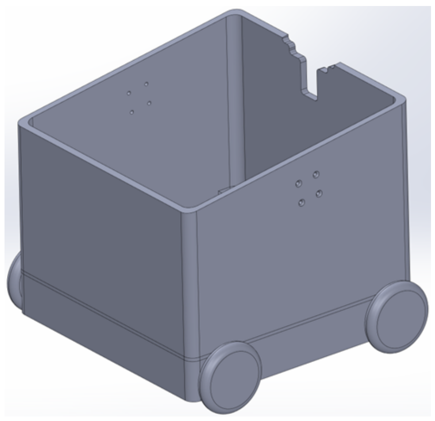
Figures 1 and 2: Exploded and isometric views of the pollen sorter, storage, and delivery system.





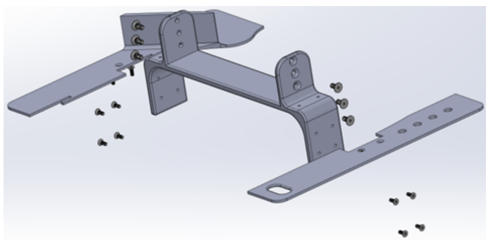
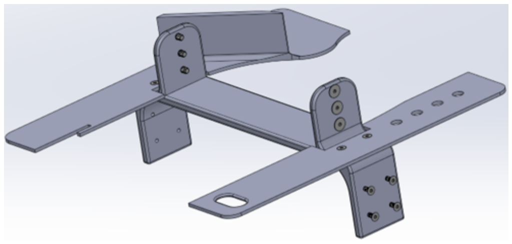
The pollen storage and delivery mechanism were the simplest design of the entire robot, consisting of one Miuzei MG996R Servo connected to a gate at the end of the storage area which was raised and lowered using a double link arm. Once the robot had traveled the desired distance and stopped, the gate servo would activate, opening the gate and dumping the pollen into the plant. Since there were no sensors in the storage area which could determine if all of the pollen had been unloaded, several tests were conducted to determine the average length of time required for the gate to stay open before all of the pollen had been unloaded.

Figures 3 and 4: Exploded and isometric views of the electronics box and drive chassis.

The electronics box and drive chassis, seen above in Figures 3 and 4, were created to ensure that the robot was as stable as possible, and to ensure that there was more than enough space for the robot's electronics to be stored in an organized manner. As seen from the exploded view in Figure 3, the electronics box and drive chassis consisted of a three-layer design. The bottom portion, the chassis, held the motors and the IMU which were then isolated by a lid. The electronics box, which also acted as a mounting point for the pollinator, was then screwed down to the lid, giving a 9.5"x9.5" space for the electrical components to be laid out in an organized manner.

Figures 5 and 6: Pollinator and display mounting.

Finally, the pollinator and display were mounted to the top of the electronics box to achieve the desired height required for pollen to be deposited to the top portion of the plant. Along with the pollinator mount, the side covers also work as a mount for the display components, namely, the LED strip indicator and the rotary encoders that control said LED. Along with the display components, the activation switch was also mounted to the cover for ease of access. While the left cover served to mount the display components, the right cover served to redirect dumped balls so that they do not interfere with the path of the robot. This mount is shown in Figures 5 and 6.

The robot is driven by 2 25SG-370CA Series DC gear motors with an integrated gear box with a gear ratio of 4:1, outputting 1.2 Kg.cm of torque at 500 RPM with an input of 12 V & 750 mA (rated). These motors are paired with two wheels of 65 mm diameter and 26 mm width and are mounted to the chassis using a mounting bracket provided in the wheel kit. The system also includes a magnetic encoder to measure the distance traveled by the wheel, which is explored further in section 3.5. Each motor encoder unit has 6 output wires, the wiring diagram enclosed in the Appendix. The direction and speed of both motors is controlled by an L298N dual H-bridge motor driver, providing the required 12 V of power to each motor. Since both motors are identical, their wiring remains the same (including wire colors), and the motor power inputs are wired to opposite ends of the motor output terminals on the H-bridge. The H-bridge inputs come from the central microprocessor in the system, including PWM and direction signals, except for the 12V power input, which comes from the battery circuit.

There are two separate power sources to the system, delivering two voltage levels to the components. The DC motors, requiring a higher voltage of 12 V are powered by a Sony VTC6 4S 3000 mAh battery that outputs 14.8V. This 14.8 V input is stepped down to 12 V, and attached to a hardware kill-switch that will turn off power to the motors, acting as an e-stop button for the motors. The other subsystems, including the central controller and all peripherals, are powered by a 10,000 mAh power bank through USB 2.0 power delivery. This allows for a 5V input at up to 500 mA, which is adequate for powering all subsystems not including the DC motors.

2.3. Computing Hardware and Inter-Device Communication

The system uses a controller-peripheral hierarchy to reduce the computational complexity of the system and limit the current draw and pin use from each individual computing device. These peripherals are connected to and powered by the main microprocessor via USB port. The sensor interface between the devices in the system is summarized in Table 1.

TABLE 1. Hardware/software hierarchy of the system.

Central Computer		Peripheral 1		Peripheral 2	
Raspberry Pi 3B (Python)		QTPY (Arduino)		QTPY (Arduino)	
Inputs	Outputs	Inputs	Outputs	Inputs	Outputs
Wheel encoder inputs x2 Start button	Motor PWM x2 Motor direction	Color sensor *Serial input*	Servos x3 *Serial output* Color data	Encoder color inputs x3	RGB LED strip (plant color and game status) *Serial output* Plant 3 color

The central controller does high-level processing of data and makes motion planning decisions, as well as performing the general control logic. It is a Raspberry Pi 3B microprocessor operating using a Debian GNU/Linux-based OS and programming is completed in Python 3. This was selected over an Arduino-exclusive central computer in order to provide compatibility with the Robot Operating System (ROS), which was initially considered in more complex navigational schemes for the robot. It can be programmed via SSH protocol and a terminal emulator like MobaXTerm (Mobatek, Toulouse, France). It interfaces with the DC gear motors, wheel encoders, and the start game button via GPIO pins. It also interfaces with two peripherals. The central controller receives serial input from the peripheral controllers and in turn sends serial commands to prompt the peripherals to perform basic functions.

The peripheral controllers are two Adafruit QTPY microcontrollers programmed using Arduino. These peripherals perform low-level tasks, with sensing divided between the two devices. Due to the small size of the QTPY microcontrollers and limited pinout, two microcontrollers were used instead of one. The first peripheral reads color sensor data, outputting a serial string representing the last color read. It receives another serial string from the central microprocessor indicating which servo motor associated with the pollen sorting and dispensing system should be actuated. The second QTPY peripheral interfaces with three rotary encoders for inputting the color of each plant and an RGB LED strip for showing the color of each plant and the game status. Using serial communication, the color of Plant 3 is relayed back to the main computer via and the QTPY accepts a string representing the game status. In the Python software running on the central microprocessor, the QTPY devices are represented using a custom Python class to succinctly perform read and write logic.

2.4. Sensor and Actuator Power and Calibration

The distance traveled by each wheel of the robot is measured using a magnetic encoder, which uses two Hall sensors offset from one another, fixed to the motor body. These sensors generate pulses caused by a rotating magnet attached to the motor shaft as the magnetic field switches, read by the encoder and sent to the Raspberry Pi as 'pulses'. The encoder is powered by a 5V DC input. This sensor is calibrated by writing code to constantly output the encoder pulses, then counting the number of pulses in one rotation of the wheel. This was found to be about 678 pulses per rotation for both encoders. This data is then used, along with the radius of the wheels, to calculate the distance traveled by each wheel, which is averaged and used to control movement states in the FSM.

The ball color is determined by the TCS34725 RGB color sensor. This board is capable of being powered by 3.3-5 V DC and is controlled by the I2C protocol. It is powered using the 3.3V pin on one of the QTPY microcontrollers. It is lit up by a 4150°K LED providing neutral illumination and has an IR filter to prevent any infrared noise. The sensor outputs 4 channels of data corresponding to the RGB color values (r, g, b) as well as clear light intensity (lux). This allows for accurate detection of ball color, minimizing the chances of the robot confusing similarly colored balls. Lux or RGB values were used to determine each color as summarized in Table 2. The rotary encoders used for taking the input for setting the plant color and display were set as shown below in Table 3. For each detent, the indicated plant color for the plant corresponding to that encoder changes between red, blue, and green, with all the encoders starting at blue and cycling through these colors with each turn.

TABLES 2 AND 3. Color sensor calibration values (left) and encoder rotation colors.

Output	Criteria
No Ball	lux < 50
Green	g > 1000
Yellow	r > 1600; lux < 20000
Red	lux > 20000
Blue	all others

Rotation	Color
Default	Blue
1	Red
2	Green
3	Blue

2.5. Motion Planning

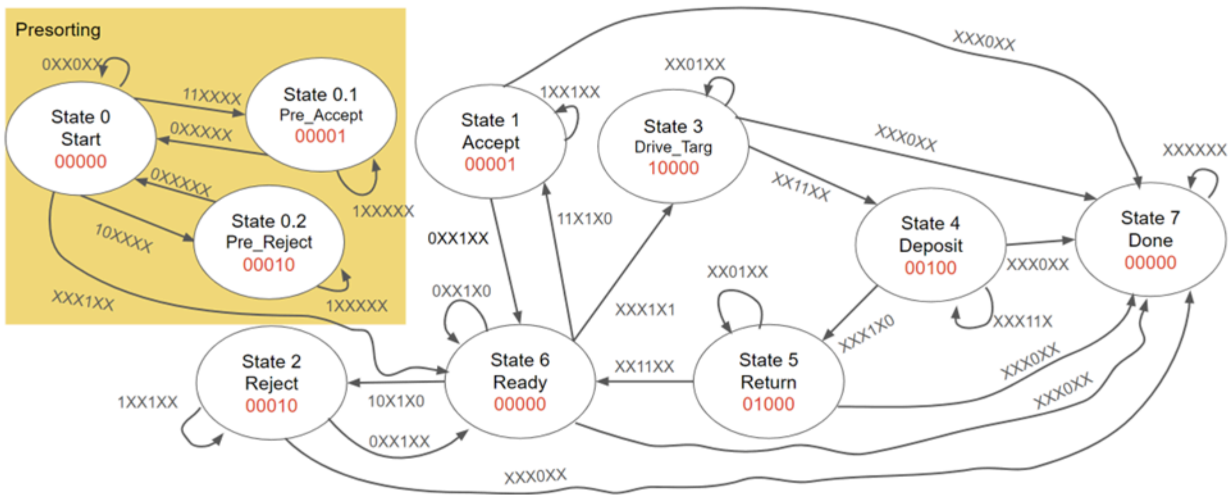
Motion planning is done using a simple closed-loop control schematic that takes in wheel encoder distance readings. As the selected game strategy only requires forward motion back and forth between two points (Plant 3 and the barn), only the x-direction distance along a horizontal line from the barn to Plant 3 is considered. At each time step, if the distance between the goal and current position (defined as the distance relative to the robot's starting position) is positive, then the motors spin forward. Otherwise, they spin backward. The speed is constant, with only directional changes. The success condition, or when the robot reaches the desired goal (either Plant 3 or the return to the barn) is defined as a Boolean value that is True when the difference between the goal and position is less than a specified value.

Initially, other motion planning schemas were considered, such as PID control and closed-loop control with both IMU and wheel encoder data. Before settling on the Plant 3 game strategy, more advanced localization using an extended Kalman filter was considered. However, the reduction of mapping complexity achieved by limiting motion to a straight line between Plant 3 and the barn made it possible to reduce controller complexity. An IMU was initially implemented in case heading information was necessary, but during testing, the robot went reliably straight, and as such, a 2D control solution was not deemed necessary.

2.6. Control Strategy

The control strategy is implemented using a finite state machine (FSM) on the central microprocessor. This FSM is updated with inputs at every time step at a frequency of 2Hz using a Python-based script. It is generally divided into three distinct phases: the pregame/presorting phase, the main gameplay phase, and the end game phase. During the presorting phase, pollen of the color corresponding to Plant 3 is accepted and added to the stores, while all other pollen colors are rejected. The robot stays in this phase until the game timer starts, as indicated by a start game button. In the main game phase, the robot moves between the barn and Plant 3, dispensing and reloading pollen when needed. The same sorting procedure as in the pregame state is employed when the robot runs out of pollen and it returns to the barn. The final phase is the end phase, with no more movement or gameplay.

Figure 7: FSM outputs and their definitions.



The control FSM employs 10 states, as shown in Figure 7. To transition between these states, there are six Boolean inputs, defined in Table 4. These states have 5 associated outputs, also summarized in Table 5. In brief, the robot starts in the Start state and moves into the Pre-Accept and Pre-Reject states and back to the start state as it receives and sorts pollen. These transitions are prompted by the sensing of pollen (PS) and color of the pollen (CS) and prompts appropriate servos to move the pollen (RP, AP). It remains in this set of states until the start game toggle button is triggered (SG). The robot then transitions to the Ready state, where it continues to accept and reject pollen in the same way as the Pre-Accept and

Pre-Reject states, until the accepted amount of pollen equals or exceeds the desired amount (FP). This cues a transition into the Drive to Target state, where the robot moves to Plant 3 via the target pattern (TP). After arriving at Plant 3, as indicated by a position measurement within a specified error tolerance of the goal which is the Plant 3 position (LR), the robot proceeds into the Dispense state, where the dispense gate servo (PD) is actuated. This then triggers a Return to the barn to refill, activating the home pattern (HP). When the LR input is again satisfied by acceptable closeness to the new goal, the home position, the robot again enters the Ready state and prepares to receive pollen. This process repeats until the timer runs out and the robot permanently enters the Done state. The timer is paused during reloading, as per game rules.

TABLE 4. FSM inputs and their definitions.

Input	Definition	Method
PS	1 if pollen is sensed at input	Current color sensor reading is different from the previous reading and is not "no ball"
CS	1 if pollen is desired color	Reading is equal to the desired color
LR	1 if movement to the goal (Plant 3 or barn) is complete	Position is within error tolerance of goal (target or home)
SG	1 if game is active	Goes to 1 when the start button is triggered and 0 when the elapsed time is > 120s
NP	1 if there is at least one pollen in the bot	Pollen count > 1; pollen is incremented each time the accept servo is actuated
FP	1 if bot is "full" and ready to move to location	Pollen count >= 5

TABLE 5. FSM outputs and their definitions.

Input	Definition	Method
TP	1 to move to target (Plant 3)	Wheels spin forward
HP	1 to move to home/barn	Wheels spin backward
PD	1 to dispense pollen in bot	Actuate dispense servo
RP	1 to reject recently sensed pollen	Actuate rotation servo and gate servo
AP	1 to accept recently sensed pollen	Actuate gate servo to allow pollen into storage container

3. Results

3.1. Motor Testing

To test motor functionality, a test code was generated, and the system was wired up to connect the motors, encoders, H-bridge and Raspberry Pi. Three tests were conducted. The first was ensuring both motors had identical PWM duty cycle speeds. To test this functionality, both motors were given identical PWM duty cycles, and the distance traveled by both wheels were measured and found to be identical. This assured us that no scaling of the PWM inputs was required between the motors and that both could be run with the same PWM duty cycle. The second test was ensuring both motors would run in identical directions. This was done by generating a code looping between two directions for both motors, and adjusting the HIGH and LOW direction GPIO outputs until both motors were traveling in the same direction (essentially causing opposite outputs, where one motor would be traveling in reverse relative to the other). Both these tests allowed us to confirm the PWM duty cycles and direction inputs of the motors.

Once the motors were calibrated and traveling in the same direction, they were tested with load by physically running the robot forward and back, to adjust the required PWM cycle to move the robot at a desirable speed. This was done by running a test code that just runs the motors in both directions, and fully wiring and assembling the robot chassis before doing so. The motors moved the robot successfully.

3.2. Color Sensor Testing

To test the color sensor, we first needed to calibrate our code. Some initial tests were done to figure out which lux values correspond to which color, and this was tested multiple times in different environments to account for ambient light. Once we had values, it was time to verify they were accurate. All four colors were tested in random order and our values were adjusted until the output was consistently accurate. The color sensor was then mounted to the top section of our chassis along with the servos. The color sensor code was tested alongside the movement of the servos to ensure the appropriate behavior. Our robot only stores one color of pollen at a time, so one color was set to be accepted, and the others rejected. This was tested multiple times across all four colors to ensure consistency.

3.3. Bidirectional Communication Testing

When testing the serial communication, we wanted to start simple to make troubleshooting easier. We started off by testing the writing of the Raspberry Pi and the reading of the QTPYs. Simple commands were sent to each of the QTPYs from the Raspberry Pi, such as moving a single servo, or changing the color of the game status LED. Once we verified these processes worked, we tried sending serial data to both QTPYs at the same time. It was then time to test the writing of the QTPYs and reading of the Raspberry Pi. In isolation, each of the QTPYs were able to communicate effectively with the Raspberry Pi. However, when both QTPYs tried to send a serial signal to the pi, one serial port would flood the Raspberry Pi with data, and it wasn't able to read the other serial port effectively. After implementing some delays between sending serial data in the code, the Raspberry Pi was able to read from both QTPYs at the same time. Reading and writing at the same time did not pose issues.

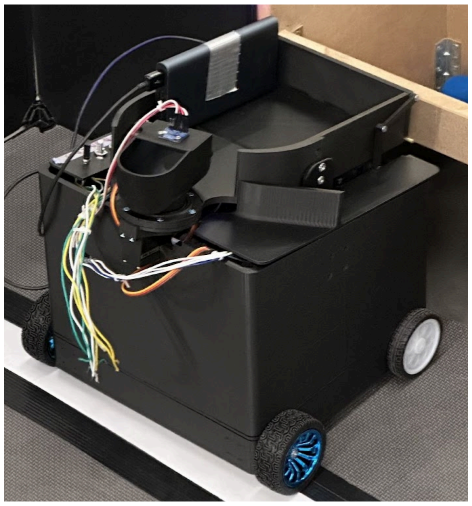
3.4. Encoder Testing

In testing the encoder functionality, test code continually measuring and outputting the encoder pulses was run. Tape was attached to the wheel and robot chassis to set a baseline in measuring one wheel rotation. The wheel was then manually rotated by hand to achieve one full rotation, and the number of pulses recorded in doing so was measured. This process was repeated 3 times for each wheel to obtain average values and eliminate any possible noise or human error, and was found to be 678 pulses per rotation for every wheel. This would then allow us to calculate the number of pulses recorded by each encoder, divide this value by the PPR (Pulses per rotation) to obtain the number of wheel revolutions, then multiply the number of revolutions by the circumference of the wheel to get the distance traveled by each wheel. Since the robot is always traveling in a straight line and the motors have identical PWM and direction signals, the distance measurements per wheel are averaged out to obtain a more accurate straight-line distance. Experimentally, this was calculated by having the robot drive a set distance (in pulses) and then using the measured distance in inches over measured pulses as a scaling factor.

4. Discussion

The robot, shown fully assembled in Figure 8, was successfully able to complete tasks required of it, despite the challenges and setbacks in its production and testing process. As discussed in the results section, most of the errors encountered in testing the system and its code were resolved through testing and playing to the strengths and limitations of the robot in its game strategy. Points of improvement in the system that were noted include adding a more complex motion planning strategy that would allow the robot to navigate its surroundings more effectively and locomote to the other plants, as well as to implement an improvement to its navigation system using the IMU which we were not able to implement on time due to component, time and code concerns.

Figure 8: Assembled robot, as viewed during checkoff.

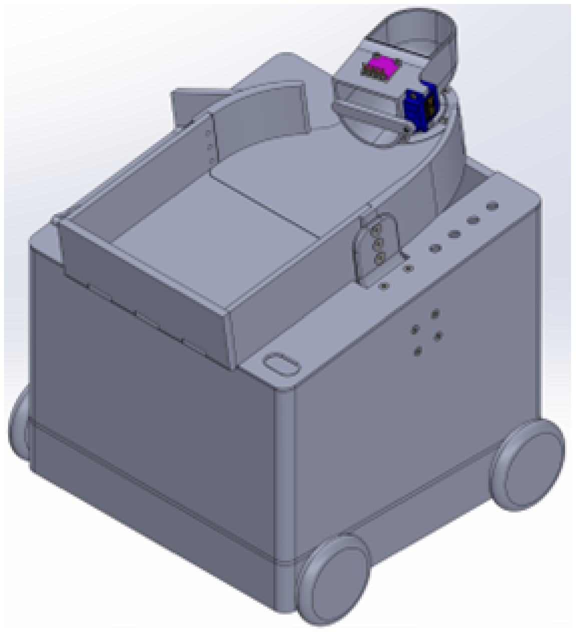
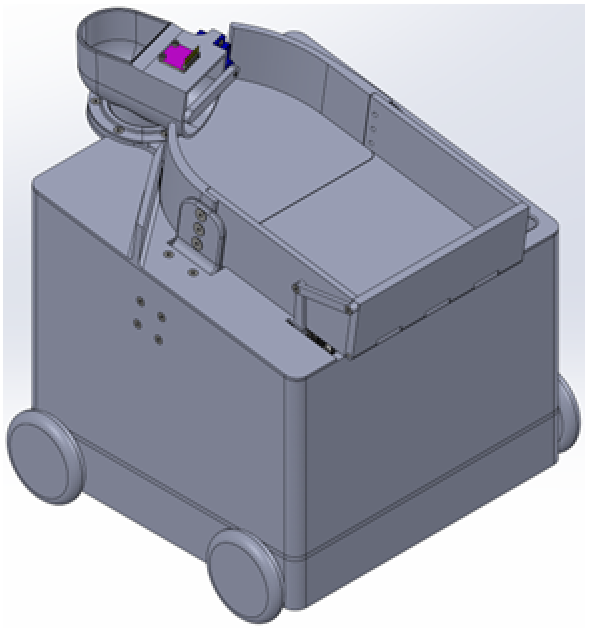


In completing this project, several constraints were encountered and solved to get the robot working. The first major constraint was caused by supply chain issues, where critical parts to the robot ordered as part of the team budget provided by the University were never delivered. This was solved by ordering parts from alternate suppliers using our personal budget, although it did delay our part testing by several weeks. The second major constraint was encountered in the final weeks of the project, where a short on one of our boards caused by a loose wire destroyed the Raspberry Pi 4 controlling our robot. This was solved by replacing the destroyed board with an alternate Raspberry Pi 3, although this did lead us to have to rewrite most of our code as the Raspberry Pi 3 does not support newer versions of Python which were required by some of the libraries used in controlling components. This was solved by rewriting most of our codebase using older libraries that were compatible with the version of Python in the Raspberry Pi 3 and retesting everything, as well as ensuring no loose wires would cause any possible shorts and replacing unsoldered connections with terminal blocks.

5. Conclusion

Using FSM programming and sensor integration, the robot met all ME588 project requirements despite significant external challenges securing a final design functionality check off. The team learned many lessons when designing a robot system, such as securing wire connections and acquiring and using reliable vendors. To create a more robust autonomous system in the future, incorporating further navigation tools such as an inertial measurement unit is suggested as relying solely on one navigation tool, in our case encoders, is prone to error. It was evident that redundancy is vital in FSM integration to ensure correct state switching. The team disassembled the robot and returned the parts to the ME department to allow their use by future semesters.

Appendix A: CAD

Figures 9 and 10: Left and right corner isometric views of the full robot assembly.

Appendix B: Electrical Schematics

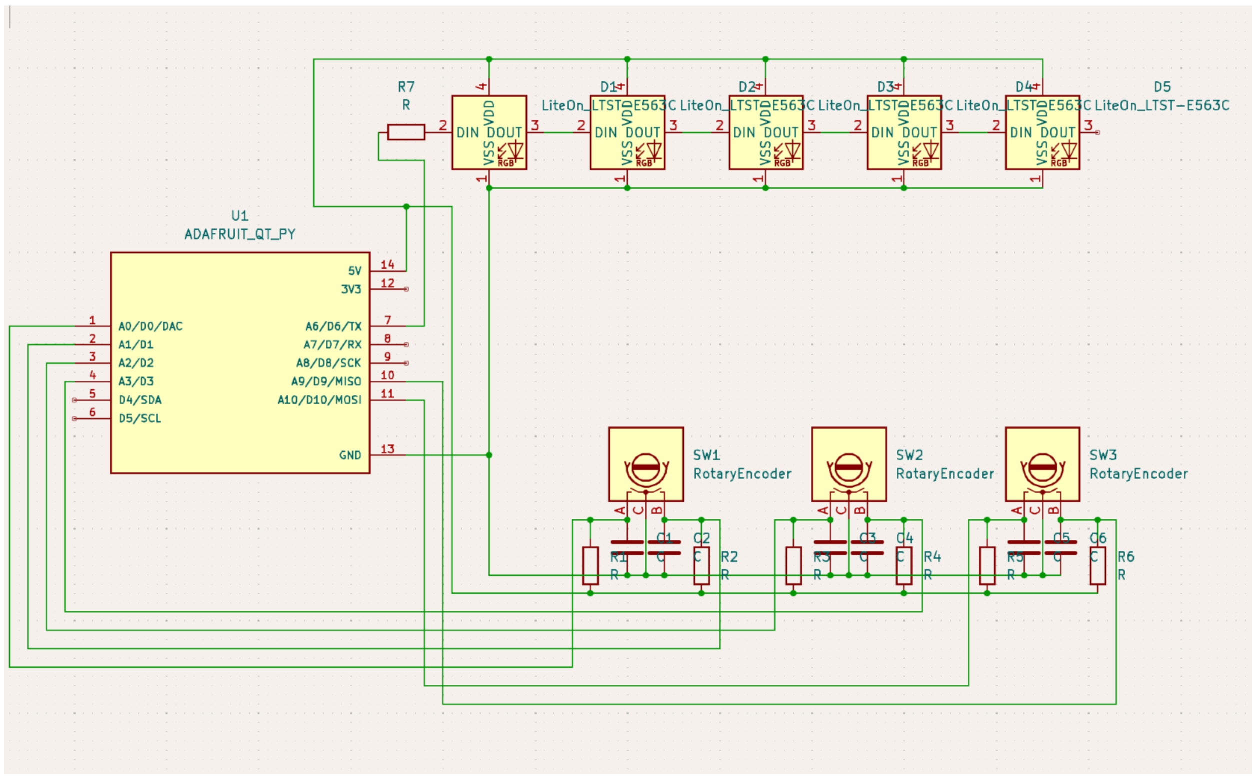
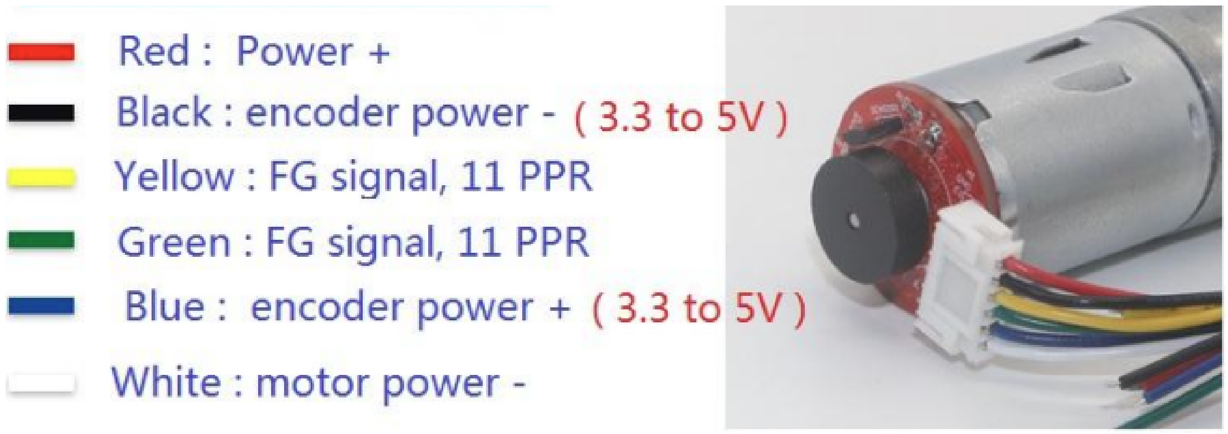


Figure 11: Circuit schematic for user inputs and display.

Figure 12: DC Motor/Encoder wiring diagram.



Red : Power +
Black : encoder power - (3.3 to 5V)
Yellow : FG signal, 11 PPR
Green : FG signal, 11 PPR
Blue : encoder power + (3.3 to 5V)
White : motor power -

Figure 13: H-bridge wiring diagram.

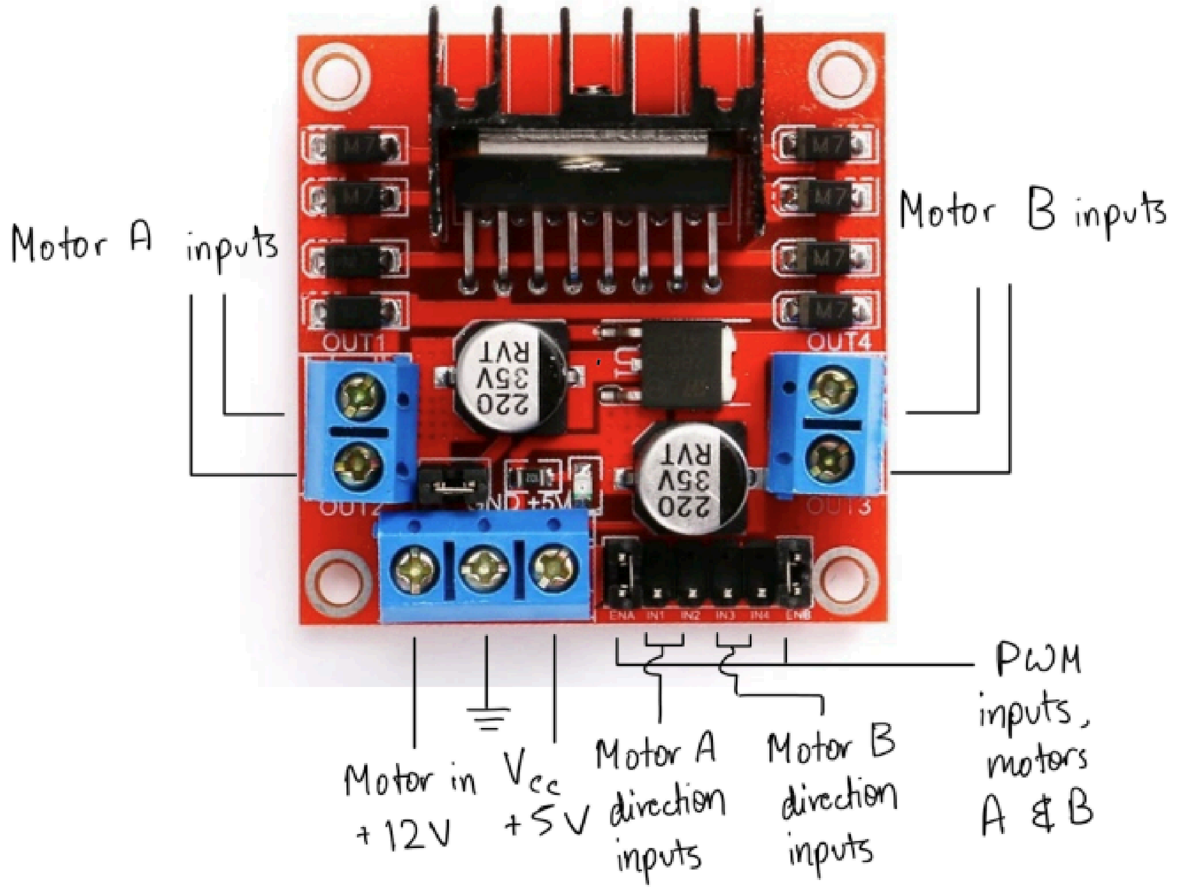


Figure 14: Power conversion circuit wiring diagram.

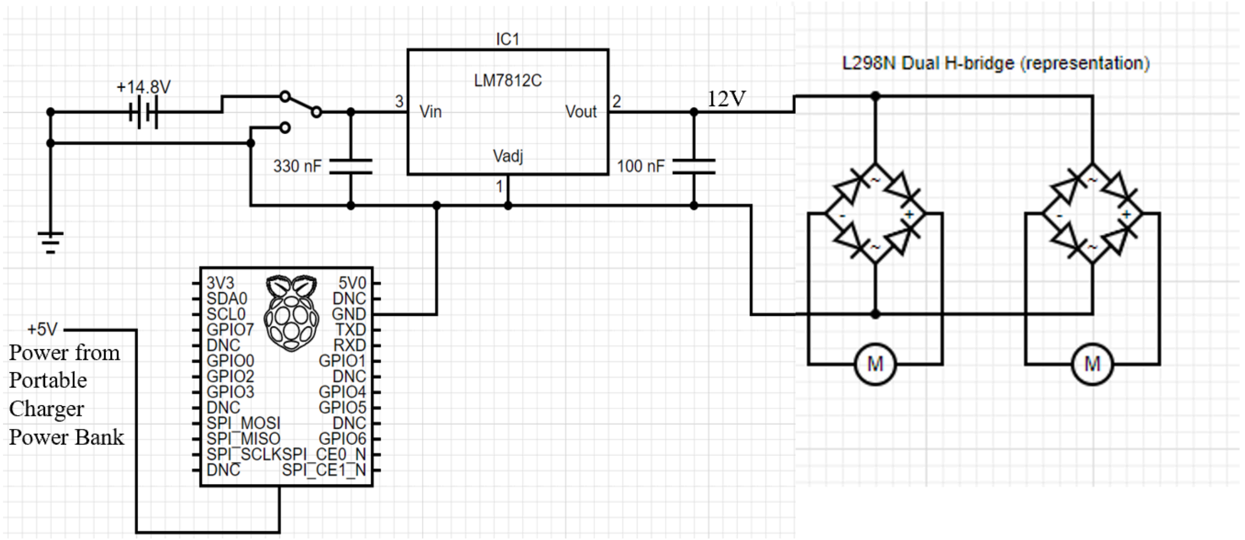
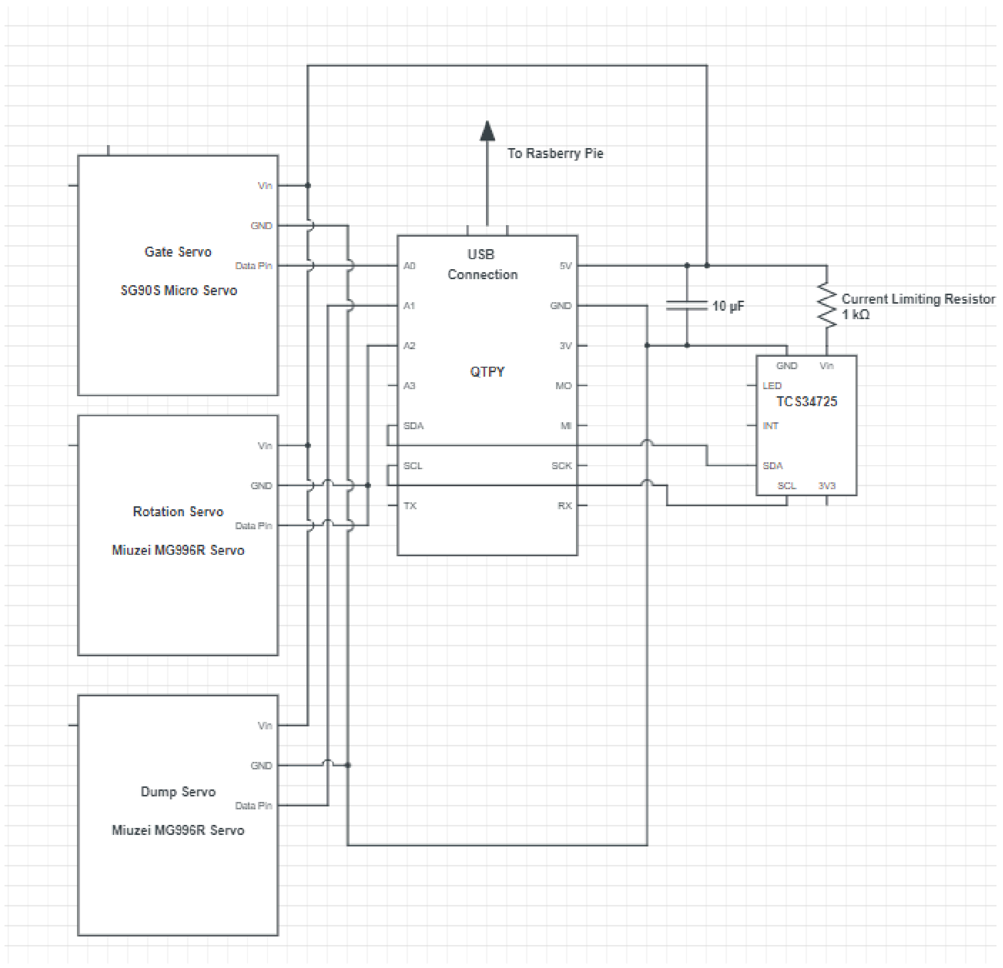


Figure 15: Pollen storage QTPY schematic.



Appendix C: Bill of Materials

Component	Quantity	Cost / $	Source
DC Motor/Encoder/Wheel	2	34.68	Personal
L298N H-Bridge	1	6.37	Personal
65mm Rubber RC Wheels	2	5.50	Personal
Arduino Uno R3	1	27.60	Personal
Rotary Encoder	3	4.50	Purdue
RGB LED	8	2.00	Purdue
Toggle Switch	2	1.25	Purdue
BNO055 IMU	1	29.95	Purdue
4-pin to Premium Male Headers Cable	1	0.95	Purdue
HC-SR04 Ultrasonic Sonar Distance Sensor	3	3.95	Purdue
Raspberry Pi 3 Model B	1	35.00	Personal
Miuzei MG996R Servo Motor Metal Gear High Speed Torque Digital Servo	1	12.99	Purdue
MG90S 9g Servo Motor Micro Metal Gear	1	13.99	Purdue
Sony VTC6 4S	4	26.16	Personal
10k mAh Power bank	1	10.00	Personal
TCS34725 Color Sensor	1	6.99	Personal