Verification & Validation

Group #12 – Botanica

SproutBot

Arun Mistry

Mina Demian

Nicholas Levantis

Usman Minhas

Table 1: Revision History

Date	Developer(s)	Change
March 28, 2024	Arun Mistry, Mina Demian, Nicholas Levantis, & Usman Minhas	Initial Verification and Validation
March 25, 2024	Arun Mistry	Added Feedback Integration table along with comments.
March 26, 2024	Usman Minhas, Mina Demian	Updated SpeedControl Review Added tests for ultrasonic sensing. Updated tests and traceability matrix.
March 27, 2024	Mina Demian	Added unit tests for motors, IR Navigation, Robot Ultrasonic, Arm Ultrasonic, Moisture Sensor, Colour Sensor and servos.
March 27, 2024	Usman Minhas	Updated unit tests for PM and PD subsystems.

Table of Contents

List of Figures

List of Tables

1. Purpose

SproutBot is a plant watering robot that aims to provide a reliable and efficient solution for ensuring the health and well-being of indoor greenery, especially when plant owners are away from home. Its primary objective is to water plants around the house as and when necessary. SproutBot will be considered a success if it is able to move, navigate towards a plant's location, and deliver water to it, repeating this process for other plants as required, and finishing its trip by returning to its base location.

SproutBot is divided into 2 main subsystems, with its movement and navigation system as one, and the watering mechanism as another. These systems will be developed separately and combined in the end. Further details are given under.

2. System Overview

2.1. Project Scope

The scope of this project is to have the robot navigate toward plants with minor adaptability (such as stopping & going around obstacles) to various plants around the same height and deliver a specific quantity of water on a schedule basis. As the potential of this project is massive, we plan to limit the scope of the project to features we believe we can implement successfully.

The scope of this project does not include:

- Plant pots of varying heights. The range of plant pot's height will be within a certain range.
- Advanced terrain navigation. The robot will not be required to climb any obstacles to navigate towards a plant. Basic movement and obstacle detection, with rudimentary obstacle avoidance will be within scope, and expanded upon as necessary.
- Elevated plants. The robot will only be required to tend to plants located on the floor of the home.
- Plants on different floors of a home. The robot will not be required to travel up and down stairs or ramps.
- Outdoor operation. This robot will be for indoor use only and will not be built resistant to different weather elements such as strong winds and rain or snow.

2.2. Project Constraints

There are 2 main constraints to keep track of when working on this project.

1. Budget. There is a maximum budget of $750 for the Bill of Materials (BOM) that should not be exceeded.
2. Time. Revision 0 is due by January 26, 2024, which must capture the major components of the project. Revision 1 is due by March 27, 2024, which must build upon Revision 0. The project's ultimate completion date is to be the date of the Final Presentation, April 26th, 2024.

3. Validation

The objective of SproutBot was set out under the Project Goals document, which will be slightly expanded upon here. The robot must be capable of autonomously navigating an indoor environment, locate houseplants, and water them. All of this must be done without user intervention. To achieve this, several different systems must be integrated together. This section will involve validating whether these systems work as expected with goals defined at a high level.

Validation ID	Goals	Components	Required Behaviour
VAL_1	Robot should be able to move at a controlled speed.	• Robot Wheels • Robot Motors	The robot must be capable of motion at a controlled speed in a 2D plane, along the floor.
VAL_2	Robot should be able to find a plant or a base.	• Robot IR Detector • Plant IR Emitter	The robot must be able to point towards a destination and get ready to move towards it.
VAL_3	Robot should be able to navigate towards a destination.	• Robot Wheels • Robot Motors • Robot IR Detector • Plant IR Emitter • Robot Ultrasonic Sensor	The robot must be capable of moving towards a destination, along with adjusting its path if it deviates from its path.
VAL_4	Robot should be able to avoid obstacles.	• Robot Ultrasonic Sensor • Robot Wheels • Robot Motors	When an obstacle is detected, the robot must be capable of stopping without colliding, turning to a direction without obstacle, and moving around it. The robot must also be able to find the destination after finding the obstacle.
VAL_5	Robot should stop when it reaches its destination.	• Robot Ultrasonic Sensor • Robot Wheels • Robot Motors	Upon reaching its destination it must slow down and stop before colliding.
VAL_6	Robot must be able to identify soil correctly.	• Robot Colour Sensor • Plant Indicator LEDs	The robot's watering arm must be able to find the soil accurately before water flows out.
VAL_7	Robot must be able to deliver water to a plant.	• Robot Water Pump	The robot's water pump must be able to pump water from the water tank to the plant's soil.
VAL_8	Plant must report the soil water levels.	• Plant Moisture Sensor	The plant must be able to measure soil moisture and report it to the robot.
VAL_9	Plant must be able to request for water.	• Plant Moisture Sensor	The plant must recognize when it is low on water and request the plant for water.

Table 1: Validation Goals and Required Behaviour

Validation ID	Current Behaviour	Future Modifications and Justifications
VAL_1	The robot is capable of moving front and back at a controlled speed. The robot is capable of turning left and right in one place at a controlled speed. The robot is able to move smoother than before when increasing speeds.	The robot's motions need to be smoothed out more, and the robot must have capabilities of moving at different speeds for the forward and backward movements compared to the speeds for turning left and right in one place.
VAL_2	The robot is able to point towards its destination by rotating its body until the IR emitter and detector line up correctly.	The IR detection is very weak right now and has limited range. An op-amp could be integrated into the detector to detect smaller values easily. A backup algorithm must be implemented in case a destination cannot be detected, in order to move around and search for an IR LED.
VAL_3	The robot is capable of moving in a straight line with minor deviations in its path. If the signal is lost, the robot turns around in one place until the signal is obtained again.	Thought needs to be put into how to accommodate for minor deviations, which could be done with another IR Detector.
VAL_4	This has not been implemented yet.	The robot must stop before hitting an obstacle using ultrasonic sensors. Obstacle avoidance algorithms must be implemented to move around the obstacle and search for the destination again.
VAL_5	The robot uses IR to get close to the destination, then switches to ultrasonic sensor near the destination. Once it reaches close enough, it stops before colliding.	The speed of movement needs to be slowed down after switching to ultrasonic sensor. The robot could employ minor rotations to confirm it is pointing in the correct direction.
VAL_6	The robot turns around in a small radius in front of the robot until it finds a blue LED indicator placed in the soil. Once it finds the blue LEDs, robot can confirm that the sprout is on top of the soil.	Modify this solution to remove blue LEDs from the soil. Consider using funnels outside pot to channel water. Consider alternate solutions to detect soil, such as ultrasonic sensors.
VAL_7	The robot turns on the pump for a certain duration.	Measure the amount of water pumped per second to know how long the pump needs to be turned on for.
VAL_8	This has not been implemented yet.	Implement functionality for plant systems to read and record the amount of water in the soil.
VAL_9	This has not been implemented yet.	Implement functionality for the plant to request the robot for water.

Table 2: Validation of Current Behaviour and Future Modifications (Continuation of Table 1)

4. Verification

The verification procedure of SproutBot involves testing the robot and its systems against the System Requirements document. The different systems of the robot will be tested individually, and then when integrated together.

4.1. Comments on Failed Tests

It is important to note that although many of the tests have "failed", they were not implemented for the sake of revision 0 as they were not necessary to meet the points in our written contract. The contract promises:

1. The robot will start at a starting base location.
2. The robot will recognize the specific plants that need watering and start moving towards one of them.
3. The robot then navigates towards the specified plant autonomously, without user input.
4. Once the robot reaches the plant, it will dispense a specific quantity of water to the specific plant.
5. After the robot has finished watering a plant, it navigates to the next plant if applicable, and repeats steps 4 and 5 for all plants.
6. After the robot is finished with watering the different plants that needed to be watered, the robot will autonomously navigate back to the starting base location.

Requirements involving user interaction such as safety requirements "notifying the user" are all currently fails since no system for user interaction has been set up for revision 0. These are all not necessary for basic function but would be nice to have in a later revision. This is common with many of the safety requirements; in the future it will be good to incorporate them however for the sake of revision 0 it was important to get the basic functions working. Going forward, the aim will be to incorporate more of these features.

4.2. Comments on Ignored Requirements

RN3 will be ignored since the robot can navigate between base and plants without keeping an update of its position in 2D space. Also, RN5 and H1 will be ignored since the robot will recognize plants needing water not by scheduling but by moisture sensors in the plant.

There is also the possibility that some of the safety and non-functional requirements get ignored in the future if the group deems them not important. For example, the sensor redundancy has been ignored for now and may never get addressed since the redundancy may be excessive when considering the low criticality of the system. This is also the case with the user verification through username and password; since this is not a safety critical system this is out of the project's scope and will likely not be incorporated in the future.

4.3. Functional Requirements- Table of Tests

These tests are run on different components of SproutBot. These tests are done upon the completion of the robot, and so can involve testing multiple requirements in one test case.

Test ID	Description	Input	Expected Behaviour	Actual Behaviour	Pass/Fail
RM.T1	The robot must be able to move along the floor in two dimensions.	Sequentially call functions moveFront, moveBack, moveLeft, moveRight.	Pass if robot goes forward, backward, rotates left, and then rotates right.	Behaves as expected.	P
RM.T2	The robot must be able to control its speed and not move at a speed greater than *MAX_SPEED*.	Using tape, mark a known distance on the ground. Change the speed value and time how long it takes to move across the known distance. (Start the robot behind the first marker to ignore the acceleration).	Pass if the time gets shorter as the speed value increases. Pass if at the maximum speed value, the calculated speed is less than *MAX_SPEED*.	Behaves as expected.	P
RM.T3	The robot must have a maximum acceleration of *MAX_ACCELERATION*.	Using tape, mark a known distance on the ground. Starting from rest, time how long it takes for the robot to move across the known distance.	Pass if the calculated acceleration is less than *MAX_ACCELERATION*.	Behaves as expected.	P
RM.T4	The robot must be able to start from and end at a fixed location, *BASE_LOCATION*.	Run a complete sequence (start at base, locate plant, travel to plant, water plant, return to base) and determine the position of the robot at the start and end of the cycle.	Pass if the difference in the start and end distance is less then 30cm.	Behaves as expected.	P
RM.T5	The robot must be able to stay idle at a fixed location, *BASE_LOCATION*.	Start the sequence without the plant outputting a need for water.	Pass if the robot remains idle at the *BASE_LOCATION*.	Behaves as expected.	P
RM.T6	The robot must be able to move on all types of indoor flooring surfaces.	Sequentially call functions moveFront, moveBack, moveLeft, moveRight on different flooring surfaces (tile, hardwood, and carpet).	Pass if robot goes forward, backward, rotates left, and then rotates right on all surfaces.	Behaves as expected on hard floor, yet to test on carpet.	F

Test ID	Description	Input	Expected Behaviour	Actual Behaviour	Pass/Fail
RM.T7	The robot must be able to move at smooth controlled speeds, while moving in all the different directions.	Start the movement sequence and view the speed of the robot using the input for RM.T2 during and after each movement state.	Pass if the robot stops when the movementState/ movement direction changes. Pass if after the stoppingDelayDuration time has passed, the the robot starts moving again at the initialSpeed. Pass if after the robot was stopped, it starts moving again at the initialSpeed. Pass if when the robot was already moving, the speed of the robot keeps increasing until it reaches the speed required.	Behaves as expected.	P

Table 3: Robot Movement Tests

Test ID	Description	Input	Expected Behaviour	Actual Behaviour	Pass/ Fail
RN.T1	The robot must be able to autonomously travel between different locations.	Run a complete sequence (start at base, locate plant, travel to plant, water plant, return to base).	Pass if the robot is able to complete this sequence without any human interference/ control of its movement.	Behaves as expected.	P
RN.T2	The robot must be able to locate the position of a plant within a distance of *MAX_RADIUS*.	Place the robot at *MAX_RADIUS* away from plant and have the plant output the IR signal with findPlant function.	Pass if the robot begins to turn looking for the source and stops turning when it is facing the plant.	Behaves as expected.	P
RN.T3	The robot must be able to keep track of its own position.	N/A	N/A	No longer required.	-
RN.T4	The robot must notify the user if a plant is not reached within a duration of MAXIMUM_ATTEMPT_TIME.	Output the IR signal with findPlant function but keep the plant out of view of the robot so it cannot find the plant.	Pass if after the timeout period, the robot notifies the user that the time out has occurred.	There is no notification sent to the user, however the robot does stop moving after the timeout period.	F

Test ID	Description	Input	Expected Behaviour	Actual Behaviour	Pass/ Fail
RN.T5	The robot must follow a schedule to start navigation towards plants.	N/A	N/A	No longer required.	-

Table 4: Robot Navigation Tests

Test ID	Description	Input	Expected Behaviour	Actual Behaviour	Pass/ Fail
PM.T1	The system must recognize how much water is being received by the plant.	Power on the plants ESP32 system to request water. Manually water plant and monitor the detected water level.	As water is delivered, the water level value from the moisture sensor should increase proportional to the water poured.	The water level value increases gradually to show the plant is receiving water. However, it lacks accuracy and has latency as the water flows.	P
PM.T2	The system must be able to save the watering information pertaining to each plant.	Power on all components ESP32s with the robot at the base location. After a certain amount of time, the plant should request water which will cause the robot to enter search mode. Verify that water level requirements/time requirements are met for the plant	The robot should begin searching for the IR once the watering constraints for the plant are met.	Behaves as expected.	P
PM.T3	The system must be able to differentiate plants.	Power on both the plant and robot's ESP32s and monitor the serial output of the dataRecieved() function.	Upon powering on, the plant will communicate with the robot. The robot will reply with a message that should contain the unique ID of the plant.	Behaves as expected.	P

Table 5: Plant Monitoring Tests

Test ID	Description	Input	Expected Behaviour	Actual Behaviour	Pass/ Fail
PD.T1	The system must be able to identify how much water is coming out of the pump.	Power on the ESP32 and call dispenseWater().	Pass: As the pump delivers water, it uses the tested flow rate to calculate the amount of water being dispensed.	Behaves as expected.	P
PD.T2	The system must hold water up to a maximum capacity of 500 mL	Physically measure the quantity of the water tank affixed to the robot	Pass: Capable of holding up to 500mL Fail: Maximum capacity is <500mL	Tank can hold the required amount of water.	P
PD.T3	The system must be able to move with a full tank of water without spillage.	Fill tank to maximum capacity. Power on robot and call functions moveFront, moveBack, moveLeft, moveRight.	The robot should be able to accelerate, decelerate and turn at reasonable to verify that the mechanical aspect of the robot can hold this amount of water.	Behaves as expected.	P
PD.T4	The system must be able to align its water delivery to the plant's soil. The system must directly target the soil when watering. The system must have a valve for water flow.	Place robot within watering distance of a target plant and call function findSoil() to locate watering location. Once the appropriate quantity of water is delivered, the pump should close off the waterflow with the built in valve.	The robot arm should extend forward, first attempt to locate an acceptable height, then get as close as possible to the object. At this point the color sensor should power on and check the color of the object, if it determines the color is not green and is brown, it should dispense water. Then it should return to its resting position.	Behaves as expected.	P
PD.T5	The system must directly target the soil when watering.	Place robot out of watering distance of a target plant and call function findSoil() to locate watering location. This is to test failure to locate the watering target.	The arm should attempt to locate an acceptable height. Since it is not in range, the robot should timeout and return to the home base.	Behaves as expected.	P
PD.T6	The system must directly target the soil when watering.	Cover all soil on the plant with something green such as green paper. Call function findSoil() to attempt to locate the dirt.	The arm should extend and find an acceptable height, then get close to the object. At this point the color sensor should power on and attempt to avoid green while locating brown dirt. Since there is no available dirt, it should return to resting position after scanning everything.	Behaves as expected.	P

Test ID	Description	Input	Expected Behaviour	Actual Behaviour	Pass/ Fail
PD.T7	The system must follow the plant's watering schedule.	Power on the ESP32 connected to the plant with the IR beacon system connected and the robot.	At every watering interval the IR LED plus the debugging LED for the beacon system should power on and communicate with the robot's ESP32 requesting water.	Behaves as expected.	P
PD.T8	The system must have a valve to allow water to flow and stop.	Place the robot within watering distance of a target plant with visible dirt and call function findSoil() and allow it to return to base after completing watering. This is to test the closing and opening of the built in pump valve.	The robot should begin with the valve closes and no water leaking. Once detected, the robot should open the valve and water the plant. Upon completion, it should close the valve and return to the base without spilling any water.	Behaves as expected.	P

Table 6: Plant Care Delivery Tests

Test ID	Description	Input	Expected Behaviour	Actual Behaviour	Pass/ Fail
E.T1	The robot must be able to detect obstacles in its path.	N/A	N/A	Currently not implemented.	F
E.T2	The robot must keep a minimum distance of MINIMUM_DISTANCE from all obstacles.	Place the robot at *MAX_RADIUS* away from plant and have the plant output the IR signal with findPlant function. Then place some obstacles in the robot's path.	Pass if the robot is able to detect and stop when it is within the specified distance threshold *MINIMUM_DISTANCE* from any object using the Ultrasonic Sensor.	Robot is able to detect when it is within the specified distance threshold *MINIMUM_DISTANCE* from any object. However, the stop before obstacles is not implemented yet.	F

Table 7: Environment Tests

Test ID	Description	Input	Expected Behaviour	Actual Behaviour	Pass/ Fail
H.T1	The system must allow the user to set a schedule for watering each plant at different times.	N/A	N/A	No longer required.	-
H.T2	There must be a way for the user to control the robot's plant watering path.	N/A	N/A	Currently not implemented.	F

Table 8: Human Tests

4.4. Safety Requirements and Non-Functional Requirements

Safety and Non-Functional Requirements will be considered separate from functional requirements. These requirements can be considered tests themselves, and thus they will be evaluated with a checklist method instead of unique testing.

4.4.1 Safety Requirements

Req ID	Description	Current Status	Pass/ Fail
SRR1 SRR3	The system must have a water-resistant casing around the robot and any electrical components.	The water tank is kept shut, but the casing around robot and components is yet to be built.	F
SRR2 SRP6	The system must ensure the water dispersal component is directly over the soil before it is able to release any water.	Robot ensures it is over soil before water is pumped.	P
SRR4 SRP8 SRE1	The robot must avoid collisions with all obstacles around it.	The robot avoids colliding with the plant at watering, however no other object avoidance is implemented.	F
SRR5	The system must identify and reroute accordingly if a plant is relocated.	The system can detect a plant even if it has been moved if it's still visible.	P
SRR6	The system must have a feedback system with sensor redundancy to ensure its position is accurate.	No feedback or redundancy system exists, and none will be implemented upon further discussions.	F
SRR7 SRR10	The system must have a rigid movement system to avoid slipping on slick surfaces or transitioning from different floor surfaces.	Rigid movement system exists however yet to try on certain floor surfaces.	F
SRR8	The system must alert the user if its expected position does not match its actual position.	No user alert system exists.	F
SRR9	The system must alert the user in the case of a collision or if a collision cannot be avoided.	No user alert system exists.	F
SRR11	The system must stop and alert the user in the case of reoccurring collisions.	No user alert system exists.	F
SRR12	The system must alert the user if it is unable to move.	No user alert system exists.	F
SRR13	The system must have multiple sensors for redundancy should one fail.	No sensor redundancy exists.	F
SRR14	The system must detect if it is upright and notify the user if it is not.	No user alert system exists.	F
SRR15	The system must stop its movement if it receives a force greater than *MAX_FORCE*.	No force detection system exists.	F
SRR16 SRP7	The system must notify the user when the battery level reaches *BATTERY_LEVEL_LOW*.	No battery level detection exists.	F
SRR17	The system must complete a test at start up to ensure all components are connected and receive power.	This can not be tested for most components as they do not have feedback mechanisms.	F
SRR18	The system will require a method for authenticating the user, such as a user/password.	No authentication system exists.	F

Req ID	Description	Current Status	Pass/Fail
SRR19	The system must operate offline using the last schedule set by the user.	Schedules will not be implemented.	-
SRP1	The system must correctly recognize the water needed by the different plants.	The system gives the same amount of water no matter what.	F
SRP2	The system must be able to correctly identify and recognize the different plants.	The system can identify which plants are which using a unique ID.	P
SRP3	The system must follow the set watering schedules for the different plants.	Schedules will not be implemented.	-
SRP4	The system must have a way to shut off the watering system at the correct time and not allow any more water than intended.	The systems pump shuts off in a timely manner meaning the water can accurately be controlled by timing.	P
SRP5	The system must have a way to determine if the watering system/pump is not working properly.	No system for this exists.	F
SRE2	The system must have multiple valves for redundancy should one fail.	There is no redundancy for controlling the water output.	F
SRE3	The system must have a way to detect if there is any water present on the ground.	No water detection system exists.	F
SRH1	The system must be big enough that it is easily visible from a distance.	The system is clearly visible within its *MAX_RADIUS*.	P
SRH2	The system must be able to notify the user if there is any water present on the ground.	No user functionality has been implemented yet.	F
SRH3	The system must use electrically insulating materials.	Most components are currently left exposed.	F
SRH4	The system must have a way for the user to monitor the results of the subsystems.	No user functionality has been implemented yet.	F
SRH5	The system must have a manual way to shut down the robot in an emergency.	No emergency stop has been implemented yet.	F

Table 9: Safety Requirements Status

4.4.2 Non-Functional Requirements Checklist

Req ID	Description	Current Status	Pass/Fail
NF1	The robot must follow watering schedules accurately.	Schedules will not be implemented.	-
NF2	The system must report reliable plant water levels.	Plant moisture sensor has not been integrated yet.	F
NF3	The system must report reliable reservoir water levels.	Calculations to track water levels through duration of pump running has not implemented yet.	F
NF4	The robot must notify users about events within a maximum duration of *MAXIMUM_RESPONSE_TIME*.	User notification system has not been implemented yet.	F
NF5	The system must be able to recognize and scale to watering a varying number of plants.	System can recognize different plants.	P
NF6	The system must ensure efficiency in its use of power from the power supply.	Motor Drivers are inefficient and cause large voltage drops. Better motor drivers must be investigated.	F
NF7	The system's various components must be easy to maintain and replace when needed.	Components can be easily swapped out.	P
NF8	The system's must be kept secure.	No login system has been implemented yet, so no user functionality exists.	F
NF9	The connection between the system interactions and the user must be stable and reliable.	No user functionality has been implemented yet.	F
NF10	The system must be capable of adapting to minor deviations in a plant's position.	Robot rotates when a signal is lost, to find the signal again.	P
NF11	The system must be capable of adapting to deviations in an obstacle position.	No obstacle avoidance has been implemented yet.	F
NF12	There must be a way that the user can monitor the system.	No user functionality has been implemented yet.	F
NF13	There must be a way that the user can obtain information about their plants.	No user functionality has been implemented yet.	F

Table 10: Non-Functional Requirements Status

4.5. Traceability Matrix

Requirement ID	Tests																										
	Robot Movement							Robot Navigation					Plant Monitoring			Plant Care Delivery								Environment and Human			
	RM.T1	RM.T2	RM.T3	RM.T4	RM.T5	RM.T6	RM.T7	RN.T1	RN.T2	RN.T3	RN.T4	RN.T5	PM.T1	PM.T2	PM.T3	PD.T1	PD.T2	PD.T3	PD.T4	PD.T5	PD.T6	PD.T7	PD.T8	E.T1	E.T2	H.T1	H.T2
RM1	X																										
RM2		X					X																				
RM3			X																								
RM4		X																									
RM5				X																							
RM6					X																						
RM7						X																					
RN1								X																			
RN2									X																		
RN3										X																	
RN4											X																
RN5												X															
PM1													X														
PM2														X													
PM3															X												
PD1																X											
PD2																	X	X									
PD3																			X								
PD4																			X	X	X						
PD5																						X					
PD6																			X				X				
E1																								X			
E2																									X		
H1																										X	
H2																											X

Table 11: Traceability Matrix

5. Evidence Beyond Testing

Our team decided that in addition to our testing, we will conduct reviews of the code written for the different subsystems and modules. Our codebase consists of three main subsystems which are the Robot itself, the Plant Base, and the Home Base, with the Robot subsystem containing the modules responsible for the Motors, Sensors, Communications as well as other Loop Functions and the main Robot module.

5.1. Robot Subsystem Code Review

5.1.1. Motors Module

5.1.1.1 Speed Control Function

```
int speedControl(int speed) {
  static unsigned long startTime = millis();
  static unsigned long stopTime = millis();
  static bool stoppingDelay = false;  // Flag to indicate whether stopping delay is active
  static bool afterDelay = false;  // Flag to indicate the end of the stopping delay

  if (millis() - startTime > speedChangeDelay) {  // Has timeout happened yet?
    startTime = millis();                         // Reset startTime
    // If not already stopped, and switching between front/back/left/right, stop robot first
    if ((currentMovementState != 0) && (currentMovementState != previousMovementState)) {
      stopTime = millis();
      stopRobot();
      stoppingDelay = true;
      afterDelay = false;  // Reset afterDelay flag
      Serial.print("Speed set to Zero. Movement Mode: ");
      Serial.println(currentMovementState);
    } else {
      if (stoppingDelay && millis() - stopTime < stoppingDelayDuration) {
        speedSmoothed = 0;  // Set speed to 0 during the delay
      } else {
        stoppingDelay = false;  // Deactivate stopping delay once the delay is over
        if (!afterDelay) {
          speedSmoothed = initialSpeed;   // Reset speedSmoothed to 200 after the delay
          afterDelay = true;  // Set afterDelay flag to true after the delay
        } else {
          speedSmoothed = (speed * speedFrac) + (speedSmoothed * (1 - speedFrac));
        }
      }
    }
    previousMovementState = currentMovementState;
  }
  return (int)speedSmoothed;
}
```

Figure 1: Code Snippet for Speed Control Function in the Motors Module of the Robot Subsystem

Purpose:	The purpose of this function is to check if the time has passed the speedChangeDelay, to restart the time, and if the movementState has changed set the speedSmoothed value to 0, and if not update the speedSmoothed that is used to control the speed of the motors accordingly.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
speedControl()	Check if the time has passed the speedChangeDelay, then restart the time and if the movementState has changed update the speedSmoothed to 0, and if not update the speedSmoothed value either to be the initialSpeed value, if it is right after the stopDelay, or by the speedFraction if not.	○ After every period of speedChangeDelay has passed, the time is restarted. ○ If the movementState changed from the previous value, speedSmoothed variable is set as 0. ○ Then if the stoppingDelayDuration time has passed, then the robot starts moving again. ○ Then if the robot was just stopped, then the speedSmoothed variable is initialized at the initialSpeed. ○ Then if the robot was already moving, the speedSmoothed variable value increases by the speedFrac variable until it reaches the speed required.	P	Mina

Table 12: Speed Control Review

5.1.1.2 Move Front Function

```
void moveFront(int speed) {
  ledcWrite(motorPwmChannel, speedControl(speed));
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, HIGH);
}
```

Figure 2: Code Snippet for Move Front Function in the Motors Module of the Robot Subsystem

Purpose:	This function sets the pins for the two sets of motors to rotate forwards at the PWM speed.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
moveFront()	Set the Motor pins for the two sets of motors (Motor 1 & Motor 2) to rotate forward.	Pins are set properly, and robot rotates forward. ○ Motor1Pin1 is set as Low. ○ Motor1Pin2 is set as High. ○ Motor2Pin1 is set as Low. ○ Motor2Pin2 is set as High.	P	Mina

Table 13: Move Front Review

5.1.1.3 Move Back Function

```
void moveBack(int speed) {
  ledcWrite(motorPwmChannel, speedControl(speed));
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, HIGH);
  digitalWrite(motor2Pin2, LOW);
}
```

Figure 3: Code Snippet for Move Back Function in the Motors Module of the Robot Subsystem

Purpose:	This function sets the pins for the two sets the pins for the two sets of motors to rotate backwards at the PWM speed.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
moveBack()	Set the Motor pins for the two sets of motors (Motor 1 & Motor 2) to rotate backward.	Pins are set properly, and robot rotates backward. ○ Motor1Pin1 is set as High. ○ Motor1Pin2 is set as Low. ○ Motor2Pin1 is set as High. ○ Motor2Pin2 is set as Low.	P	Mina

Table 14: Move Back Review

5.1.1.4 Move Left Function

```
void moveLeft(int speed) {
  ledcWrite(motorPwmChannel, speedControl(speed));
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, HIGH);
}
```

Figure 4: Code Snippet for Move Left Function in the Motors Module of the Robot Subsystem

Purpose:	This function sets the pins for the two sets of motors to rotate in opposite direction at the PWM speed, so that the robot can rotate left.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
moveLeft()	Set the Motor pins for the two sets of motors (Motor 1 & Motor 2) to rotate in opposite directions, so that the robot can rotate left.	Pins are set properly, and robot rotates left. ○ Motor1Pin1 is set as High. ○ Motor1Pin2 is set as Low. ○ Motor2Pin1 is set as Low. ○ Motor2Pin2 is set as High.	P	Mina

Table 15: Move Left Review

5.1.1.5 Move Right Function

```
void moveRight(int speed) {
  ledcWrite(motorPwmChannel, speedControl(speed));
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);
  digitalWrite(motor2Pin1, HIGH);
  digitalWrite(motor2Pin2, LOW);
}
```

Figure 5: Code Snippet for Move Right Function in the Motors Module of the Robot Subsystem

Purpose:	This function sets the pins for the two sets of motors to rotate in opposite direction at the PWM speed, so that the robot can rotate right.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
moveRight()	Set the Motor pins for the two sets of motors (Motor 1 & Motor 2) to rotate in opposite directions, so that the robot can rotate right.	Pins are set properly, and robot rotates right. o Motor1Pin1 is set as Low. o Motor1Pin2 is set as High. o Motor2Pin1 is set as High. o Motor2Pin2 is set as Low.	P	Mina

Table 16: Move Right Review

5.1.1.6 Stop Robot Function

```
// Motor functions
void stopRobot() {
  ledcWrite(motorPwmChannel, 0);
  turnDirection = 1;
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  digitalWrite(motor2Pin1, LOW);
  digitalWrite(motor2Pin2, LOW);
}
```

Figure 6: Code Snippet for Stop Robot Function in the Motors Module of the Robot Subsystem

Purpose:	The purpose of this function is to set the pins of the two sets of motors to stop.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
stopRobot()	Set the Motor pins for the two sets of motors (Motor 1 & Motor 2) to stop.	Pins are set properly, and robot stops. o Motor1Pin1 is set as Low. o Motor1Pin2 is set as Low. o Motor2Pin1 is set as Low. o Motor2Pin2 is set as Low.	P	Mina

Table 17: Stop Robot Review

5.1.1.7 Move Wiggle Function

```
// Rotate in one direction, then another direction
// Has to follow stopRobot() before other movements
void moveWiggle(int rotateSpeed, int rotateTime) {
  static unsigned long totalTime;  // How long has robot wiggled
  if (turnDirection % 2) {         // Check Left direction first
    moveLeft(rotateSpeed);
  } else {
    moveRight(rotateSpeed);
  }
  if ((millis() - totalTime) > (rotateTime * turnDirection)) {
    Serial.println("Direction Changed when wiggling");
    speedSmoothed = 0;
    totalTime = millis();
    turnDirection++;
  }
}
```

Figure 7: Code Snippet for Move Wiggle Function in the Motors Module of the Robot Subsystem

Purpose:	The purpose of this function is to check the turnDirection, to call the moveLeft() or moveRight() functions, and then change direction after the rotation time.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
moveWiggle()	Check the turnDirection and call the moveLeft() or moveRight() functions. Then, change direction after the rotation time.	○ Depending on turnDirection, moveLeft() is called when it is 1, and moveRight() is called when it is 2. ○ The turnDirection is changed after the rotateTime is over.	P	Mina

Table 18: Move Wiggle Review

5.1.2. Sensors Module

5.1.2.1 Locate IR Source Function

Purpose:	The purpose of this function is to move through the different states for locating the IR source.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
locateIrSource()	Go through the different states (TIMEOUT_START, TIMEOUT_WAIT, IR_DETECT, IR_SUCCESS, and TIMEOUT_END) for locating the IR Source.	○ Successfully goes through different states (TIMEOUT_START, TIMEOUT_WAIT, IR_DETECT, IR_SUCCESS, and TIMEOUT_END) for locating the IR Source based on the conditions.	P	Mina

Table 19: Locate IR Review

5.1.2.2 Move to IR Source Function

```
// Move towards IR source. Switch to ultrasound when close enough to plant.
// Returns 0 if not yet reached, 1 if reached, 2 if lost.
int moveToIrSource() {
  static int irCount = 0;  // Number of times unique ultrasound was detected when confirming
  int middleIrStrength = analogRead(middleIr);

  if (middleIrStrength <= destIrThreshold) {
    stopRobot();
    irCount++;                    // Increment number of times of detection
    if (irCount >= checkIrNum) {  // Enough Ultrasound samples detected?
      Serial.println("IR Sensor Close enough, moveToIrSource complete");
      irCount = 0;  // Reset usCount
      return 1;     // Return success
    }
  } else if (middleIrStrength == minIrStrength) {  // IR Signal has been lost
    return 2;
  } else {
    irCount = 0;                  // Reset irCount if signal is found, for future function calls
    moveFront(robotIrMoveSpeed);  // Move towards IR Source
  }
  return 0;
}
```

Figure 8: Code Snippet for Move to IR Source Function in the Sensors Module of the Robot Subsystem

Purpose:	The purpose of this function is to check the IR signal while moving towards source, once detected. It determines has reached the IR source or not, or if the signal was lost.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
moveToIrSource()	Checks the IR signal while moving to determine if it has reached the IR source or not, if it is close enough to the IR source (switches to ultrasonic sensor) or if the signal was lost.	○ It can successfully determine when it has still not reached the IR source, when it is close enough to the IR source (to switch to ultrasonic sensor) and when the IR signal is lost.	P	Mina

Table 20: Move to IR Review

5.1.2.3 Ultrasonic Distance Function

```
// Return Ultrasonic Time
long ultrasonicDistance() {
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Reads the echoPin, returns sound travel time in microseconds
  long duration = pulseIn(echoPin, HIGH);
  delay(50);
  return duration;
}
```

Figure 9: Code Snippet for Ultrasonic Distance Function in the Sensors Module of the Robot Subsystem

Purpose:	The purpose of this function is to determine the Ultrasonic Distance using the time.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
ultrasonicDistance()	Sets the trigger pin for the Ultrasonic to High for 10 microseconds, then reads the Ultrasonic sensor's echo pin to determine the duration of travel.	○ It successfully sets the trigger pin of the Ultrasonic sensor and gives the duration of travel, by reading the echo pin of the Ultrasonic sensor.	P	Mina

Table 21: Ultrasonic Distance Review

5.1.2.4 Move to Ultrasonic Source Function

```
// IR close enough, switch to ultrasonic sensor now
int moveToUsSource() {
  static int usCount = 0;  // Number of times unique ultrasound was detected when confirming
  int usStrength = ultrasonicDistance();

  if (!usStrength) {
    return 0;  // Something went wrong
  }
  if (usStrength <= destUsThreshold) {
    stopRobot();
    usCount++;                    // Increment number of times of detection
    if (usCount >= checkUsNum) {  // Enough Ultrasound samples detected?
      Serial.println("Destination found by moveToUsSource");
      usCount = 0;  // Reset usCount
      return 1;     // Return success
    }
  } else {
    usCount = 0;
    moveFront(robotUsMoveSpeed);
  }
  return 0;
}
```

Figure 10: Code Snippet for Move to Ultrasonic Source Function in the Sensors Module of the Robot Subsystem

Purpose:	The purpose of this function is to check the Ultrasonic signal while moving towards an object, and to determine if it can no longer sense an object, if it has reached the threshold distance away from the object (destUsThreshold), and if it can still sense the object, but it is still far away from it, so it can continue moving forward.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
moveToUsSource()	Checks the Ultrasonic signal while moving to determine if it can no longer sense an object, if it has reached the threshold distance away from the object (destUsThreshold), or if it can still sense the object, but it is still far away from it, so it can continue moving forward at a lower speed (robotUsMoveSpeed).	○ It can successfully determine when the ultrasonic signal is lost, when it reached a point where it is close enough to the object (destUsThreshold) and when it has still not reached the object it detected and continues to move forward at a lower speed (robotUsMoveSpeed).	P	Mina

Table 22: Ultrasonic Move Review

5.1.2.5 Find Colour Function

```
// Detect Colour by rotating robot left and right
// timeout controls how long robot rotates before giving up
// Return 0 if not found yet, 1 if found, 2 if timeout
int findColour(int speed, int timeout) {
  // delay(5000);
  // return 1;  // REMOVE THIS

  int frequency = pulseIn(colourPin, LOW);
  if (frequency < blueSensitivity) {  // Blue Colour found
    stopRobot();
    digitalWrite(pumpPin, HIGH);
    delay(5000);
    digitalWrite(pumpPin, LOW);
    return 1;
  } else {
    moveWiggle(findBlueRotateSpeed, timeForOneDirectionBlue);
  }
  return 0;
}
```

Figure 11: Code Snippet for Find Colour Function in the Sensors Module of the Robot Subsystem

Purpose:	The purpose of this function is to check for the blue light inside the plant pot, once the robot has reached close to the plant pot, and if it can't see the blue light to rotate, and once it sees it turns ON the water pump.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
findColour()	Checks for the blue colour frequency using the colour sensor, and if it sees it, it starts the water pump, however if it does not see the blue light then it rotates.	○ It can successfully sense the blue lights, and starts the water pump, and when it is not able to sense the blue light, it calls the moveWiggle() function until it finds it.	P	Mina

Table 23: Find Colour Review

5.1.3. Communication Module

5.1.3.1 Setup Function

```
// Setup ESP-NOW Communication
void espNowSetup() {
  // Init ESP-NOW
  WiFi.mode(WIFI_STA);  // Set device as a Wi-Fi Station
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    ESP.restart();
  } else {
    Serial.println("ESP-NOW Initialized");
    esp_now_register_send_cb(onDataSent);  // Register Data Send Callback
  }

  // Register peer
  esp_now_peer_info_t peerInfo = {};
  memcpy(&peerInfo.peer_addr, broadcastAddress, 6);
  if (!esp_now_is_peer_exist(broadcastAddress)) {
    if (esp_now_add_peer(&peerInfo) != ESP_OK) {
      Serial.println("Failed to add peer");
    }
  }
}
```

Figure 12: Code Snippet for Setup Function in Robot Communication Module

Purpose:	This function is responsible for setting up all communication for the robot and registering peers (plants & base).			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
espNowSetup()	Setup wireless communication and callback function for confirming data send and receive. Check if peer already exists and if it does not exist then add it.	○ Sets up wireless communication and registers callback. ○ Adds peer if it does not exist already.	P	Usman

Table 24: Communication Setup Review

5.1.3.2 Send Message Function

```
// Send a message to other boards.
// toNum is 0 for Base and Robot
void sendEspNowMsg(char toMsg, char toNum, bool irValue, bool ledValue)
{
  // Set values to send
  msgData.from[0] = 'R';
  msgData.from[1] = '0';
  msgData.to[0] = toMsg;
  msgData.to[1] = toNum;
  msgData.irMsg = irValue;
  msgData.ledMsg = ledValue;

  // Send message via ESP-NOW
  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)&msgData, sizeof(msgD

  if (result != ESP_OK) {
    Serial.println("Error sending the data");
    esp_now_send(broadcastAddress, (uint8_t *)&msgData, sizeof(msgData));
  }
}
```

Figure 13: Code Snippet for Send Message Function in Robot Communication Module

Purpose:	This function is designed to send a message from the robot to either a plant or the home base.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
sendEspNowMsg()	Format message to be sent with function parameters. Send message and check result, notify of any errors.	○ The message is properly configured. ○ Message is sent and its response is verified. ○ If there is an error, it notified the user via the serial output. ○ Suggestion: Monitor the success of the second message attempt as well.	P	Usman

Table 25: Send Message Review

5.1.4. Loop Functions Module

5.1.4.1 Wait for Signal Function

Purpose:	The purpose of this function is to move through the different states of waiting for a signal.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
waitForSignal()	Go through the different states (TIMEOUT_START, TIMEOUT_WAIT, and TIMEOUT_END) of waiting for a signal.	○ Successfully goes through the different states (TIMEOUT_START, TIMEOUT_WAIT, and TIMEOUT_END) of waiting for a signal based on the conditions.	P	Mina

Table 26: Wait for Signal Review

5.1.4.2 Find Plant Function

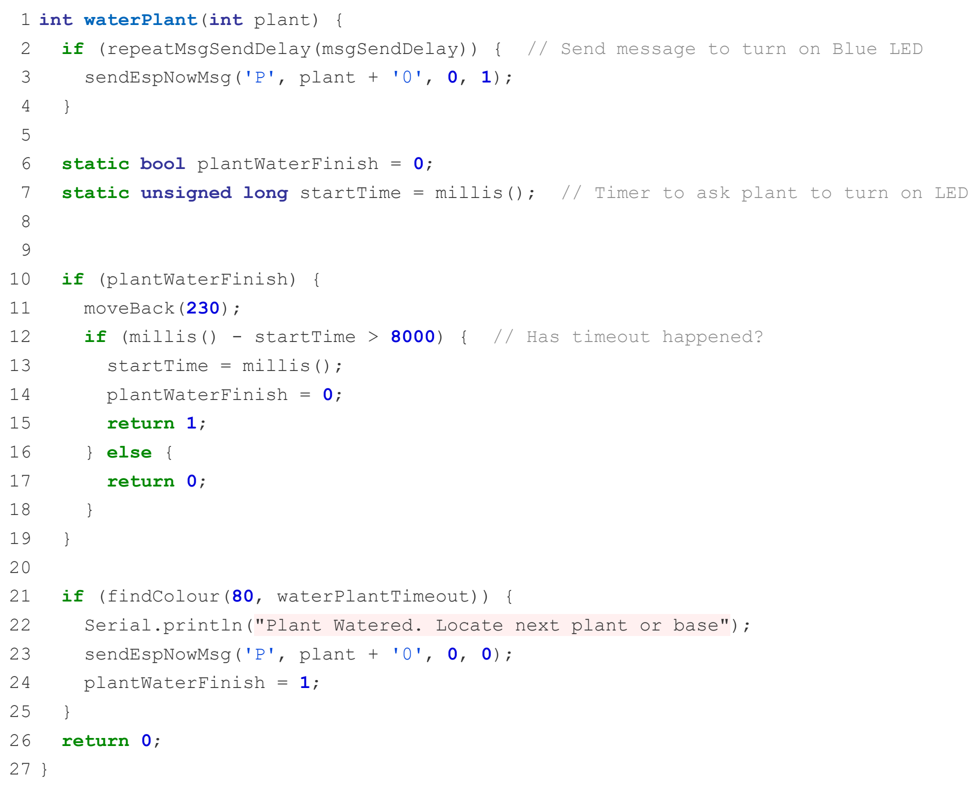
```
// Turn IR on at plant and find its location
int findPlant(int plant) {
  if (repeatMsgSendDelay(msgSendDelay)) {  // Send message to turn on LED
    sendEspNowMsg('P', plant + '0', 1, 0);
  }

  int irStatus = locateIrSource(findPlantTimeout);  // Find the IR Signal of plant
  if (irStatus == 2) {
    // TODO: Timeout Happened, Process later
    Serial.println("TIMEOUT: Plant Not Found");
    return 0;
  } else if (irStatus == 1) {
    Serial.println("findPlant Succeeded");
    return 1;
  }
  return 0;
}
```

Figure 14: Code Snippet for Find Plant Function in the Loop Functions Module of the Robot Subsystem

Purpose:	The purpose of this function is to send the message to turn ON the IR signal at the Plant location and find its location.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
findPlant()	Sends the message that turns ON the IR signal at the Plant location and then finds its location.	○ It successfully sends the message that turns ON the IR at the Plant location. ○ It can successfully find the location of the IR.	P	Mina

Table 27: Find Plant Review

5.1.4.3 Water Plant Function



```
int waterPlant(int plant) {
  if (repeatMsgSendDelay(msgSendDelay)) {  // Send message to turn on Blue LED
    sendEspNowMsg('P', plant + '0', 0, 1);
  }

  static bool plantWaterFinish = 0;
  static unsigned long startTime = millis();  // Timer to ask plant to turn on LED


  if (plantWaterFinish) {
    moveBack(230);
    if (millis() - startTime > 8000) {  // Has timeout happened?
      startTime = millis();
      plantWaterFinish = 0;
      return 1;
    } else {
      return 0;
    }
  }

  if (findColour(80, waterPlantTimeout)) {
    Serial.println("Plant Watered. Locate next plant or base");
    sendEspNowMsg('P', plant + '0', 0, 0);
    plantWaterFinish = 1;
  }
  return 0;
}
```

Figure 15: Code Snippet for Water Plant Function in the Loop Functions Module of the Robot Subsystem

Purpose:	The purpose of this function is to send the message to turn ON the Blue LEDs in the plant pot, make sure the plant is watered, and that the Robot moves back once a plant is watered, then that it looks for the next plant or base.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
waterPlant()	Sends a message to turn ON the Blue LEDs in the plant pot, and makes sure that the plant is watered, then moves back the robot, and starts looking for the next plant or base.	○ It successfully sends the message to turn ON the Blue LEDs in the plant pot. ○ It can successfully know when a plant has been watered. ○ It successfully moves back the robot after a plant has been watered. ○ Starts looking for the next plant or base.	P	Mina

Table 28: Water Plant Review

5.1.4.4 Find Base Function

```
// Turn IR on at base and find its location
// TODO: Add in Communication
int findBase() {
  if (repeatMsgSendDelay(msgSendDelay)) {  // Send message to turn on Base LED
    sendEspNowMsg('B', '0', 1, 0);
  }

  int irStatus = locateIrSource(findBaseTimeout);  // Find the IR Signal of base
  if (irStatus == 2) {
    // TODO: Timeout Happened, Process later
    Serial.println("TIMEOUT: BASE Not Found");
    return 0;
  } else if (irStatus == 1) {
    Serial.println("BASE Found");
    return 1;
  }
  return 0;
}
```

Figure 16: Code Snippet for Find Base Function in the Loop Functions Module of the Robot Subsystem

Purpose:	The purpose of this function is to send the message to turn ON the IR signal at the Base location and find its location.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
findBase()	Sends the message that turns ON the IR signal at the Base location and then finds its location.	○ It successfully sends the message that turns ON the IR at the Base location. ○ It can successfully find the location of the IR.	P	Mina

Table 29: Find Base Review

5.1.5. Main Robot Module

5.1.5.1 Setup Function

```
void setup() {
  Serial.begin(115200);  // Start Serial Comm & Logging
  motorSetup();          // Setup motor pins and PWM
  espNowSetup();         // Setup Wifi & ESP-NOW
  ultrasonicSetup();     // Ultrasonic Sensor Setup
  waterSystemSetup();    // Colour Sensor and Pump Setup
}
```

Figure 17: Code Snippet for Setup Function in the Main Robot Module of the Robot Subsystem

Purpose:	The purpose of this function is to call all the setup functions and start serial communication for logging.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
setup()	Starts the serial communication for logging and call all the Robot setup functions.	○ The serial communication for logging starts. ○ All robot setup functions are called, to set up the pins used for the motors, sensors, watering system, etc.	P	Mina

Table 30: Robot Setup Review

5.1.5.2 Loop Function

Purpose:	The purpose of this function is to loop through the different states of the system.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
loop()	Go through the different states (WAIT, FIND_PLANT, MOVE_PLANT, MOVE_CLOSER_TO_PLANT, WATER_PLANT, FIND_BASE, MOVE_BASE, and MOVE_CLOSER_TO_BASE) of the system.	○ Successfully goes through the different states (WAIT, FIND_PLANT, MOVE_PLANT, MOVE_CLOSER_TO_PLANT, WATER_PLANT, FIND_BASE, MOVE_BASE, and MOVE_CLOSER_TO_BASE) of the system based on the conditions.	P	Mina

Table 31: Wait for Signal Review

5.2. Plant Base Subsystem Code Review

5.2.1 Plant Module

5.2.1.1 Setup Function

```
void setup() {
  Serial.begin(115200);       // Initialize Serial Monitor
  pinMode(irPin, OUTPUT);     // Set IR Pin as Output
  digitalWrite(irPin, LOW);   // Turn off IR initially
  // pinMode(2, OUTPUT);       // Set LED Pin as Output

  // Init ESP-NOW
  WiFi.mode(WIFI_STA);  // Set device as a Wi-Fi Station
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    ESP.restart();
  } else {
    Serial.println("ESP-NOW Initialized");
    esp_now_register_recv_cb(dataReceived);  // Register callback if init success
  }
}
```

Figure 18: Code Snippet for Setup Function of Plant Subsystem

Purpose:	The purpose of this function is to initialize all the pins controlled by the plant and set up wireless communication between the ESP32s			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
setup()	Initialize pin for IR LED. Setup wireless communication and callback function.	○ The IR pin is initialized and set with an initial value. ○ Wireless communication is correctly set up with proper error handling and registering of callback function.	P	Usman

Table 32: Plant Setup Review

5.2.1.2 Data Received Function

```
// Callback function executed when data is received
void dataReceived(const uint8_t *mac, const uint8_t *incomingData, int len) {
  memcpy(&msgData, incomingData, sizeof(msgData));
  Serial.println("Message Received");
  lastMsgTime = millis();

  // Message is for the specific plant
  if (msgData.to[0] == 'P' && msgData.to[1] == (plantId + '0')) {
    Serial.println(msgData.irMsg);
    digitalWrite(irPin, msgData.irMsg);
  } else {
    digitalWrite(irPin, LOW);
  }
}
```

Figure 19: Code Snippet for Data Received Function in Plant Subsystem

Purpose:	This function is responsible for handling incoming communication to the plant.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
dataReceived()	Copy the incoming message and verify that it is intended for this specific plant. Complete actions requested by the communication.	○ Message is copied locally. ○ Plant ID is checked to ensure the message is intended for this plant. ○ Appropriately sets IR LED pin based on request.	P	Usman

Table 33: Data Received Callback Review

5.3. Home Base Subsystem Code Review

5.3.1 Base Module

5.3.1.1 Setup Function

```
void setup() {
  Serial.begin(115200);     // Initialize Serial Monitor
  pinMode(irPin, OUTPUT);   // Set IR Sensor Pin as Output


  // Init ESP-NOW
  WiFi.mode(WIFI_STA);  // Set device as a Wi-Fi Station
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    ESP.restart();
  } else {
    Serial.println("ESP-NOW Initialized");
    esp_now_register_recv_cb(dataReceived);  // Register Data Receive Callback
  }
}
```

Figure 20: Code Snippet for Setup Function in Base System

Purpose:	The purpose of this function is to initialize all the pins controlled by the base and set up wireless communication between the ESP32s			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
setup()	Initialize pin for IR LED. Setup wireless communication and callback function.	○ Pin for IR LED is correctly initialized. ○ Wireless communication is correctly set up with proper error handling and registering of callback function.	P	Usman

Table 34: Base System Setup Review

5.3.1.2 Data Received Function

```
// callback function that will be executed when data is received
void dataReceived(const uint8_t *mac, const uint8_t *incomingData, int len) {
  memcpy(&msgData, incomingData, sizeof(msgData));
  Serial.println("Message Received");
  lastMsgTime = millis();

  // Message is for base
  if (msgData.to[0] == 'B' && msgData.to[1] == '0') {
    digitalWrite(irPin, msgData.irMsg);
  } else {
    digitalWrite(irPin, LOW);
  }
}
```

Figure 21: Code Snippet for Data Received Function in Base Subsystem

Purpose:	This function is responsible for handling incoming communication to the base.			
Function	**Code Requirements**	**Result of Review**	**P/F**	**Reviewed By**
dataReceived()	Copy the incoming message and verify that it is intended for the base. Complete actions requested by the communication.	○ Message is copied locally. ○ Message is checked to ensure it is intended for the base. ○ Appropriately sets IR LED.	P	Usman

Table 35: Data Received Callback Review

6. Low Level Unit Tests – Table of Tests

These tests are run on the different individual software and hardware components of SproutBot before integration with other modules.

Robot Movement (RM) Unit Tests					
Test ID	Description	Input	Expected Behaviour	Actual Behaviour	Pass /Fail
RM.UT 1 (S)	The software must be able to control the increasing speed, the direction, and the movementState of the motors.	Run the movement sequence software functions, and using Serial Communication, print the speed, direction and movementState the software goes through.	Pass if the movementState, direction and speeds printed correspond to the function called, and the speed printed is increasing.	Behaves as expected.	P
RM.UT 2 (H)	The motors and wheels must be able rotate at different speeds depending on the power provided through the motor driver.	Connect the SoulBay AC/DC Power Supply with Adjustable Output to the Motor Driver and connect the motors to the motor driver, then adjust the output of the Power Supply.	Pass if the motors and wheels rotate at different speeds with the different voltages.	Behaves as expected.	P
Robot Navigation (RN) Unit Tests					
Test ID	Description	Input	Expected Behaviour	Actual Behaviour	Pass /Fail
RN.UT 1 (S)	The software must be able to detect the distance from the IR Signal.	Run the IR distance software function, and using Serial Communication, print the distance, as you move the closer and further away.	Pass if the printed distance from the IR Signal in the software is corresponding to how far the robot is from the IR Signal.	Behaves as expected.	P
RN.UT 2 (H)	The IR Emitter must be emitting a signal when powered ON.	Connect a 6V battery to the IR Signal Emitter and using a Phone Camera, check if the IR Signal Emitter is ON.	Pass if you can see that the IR Signal Emitter is ON.	Behaves as expected.	P

RN.UT 3 (S)	The software must be able to detect the distance using the Ultrasonic Signal.	Run the Ultrasonic distance software function, and using Serial Communication, print the distance, as you move the closer and further away from an obstacle.	Pass if the printed distance using the Ultrasonic Sensor in the software is corresponding to how far the robot is from the IR Signal.	Behaves as expected.	P
Plant Monitoring (PM) Unit Tests					
Test ID	Description	Input	Expected Behaviour	Actual Behaviour	Pass /Fail
PM.UT 1 (S)	The software must be able to detect the water levels using the Moisture Sensor.	Run the Water Level Detection software function, and using Serial Communication, print the moisture values, as you add more water.	Pass if the water level from the Moisture Sensor in the software is increasing corresponding to the amount of water being added.	Behaves as expected.	P
Plant Delivery (PD) Unit Tests					
Test ID	Description	Input	Expected Behaviour	Actual Behaviour	Pass /Fail
PD.UT 1 (S)	The software must be able to detect the distance using the Ultrasonic Signal.	Run the Ultrasonic distance software function, and using Serial Communication, print the distance, as you move the sensor closer and further away from an object.	Pass if the printed distance using the Ultrasonic Sensor in the software is corresponding to how far the arm is from an object.	Behaves as expected.	P
PD.UT 2 (S)	The software must be able to detect the Color of an object using the Color Sensor.	Run the Color Sensing software function, and using Serial Communication, print the RGB output, as you move objects with different colours in front of the color sensor.	Pass if the printed color value using the Color Sensor in the software is changing corresponding to the colors of the different objects.	Behaves as expected.	P
PD.UT 3 (S)	The software must be able to control the speed.	Run the servo moving software function moveServo() with a specific angle passed in.	Pass if the servo slowly and gradually gets to the desired angle.	Behaves as expected.	P
PD.UT 4 (H)	The servos must be able rotate at different speeds and directions depending on the power and input provided to them.	Manually move the servo to a resting position. Connect a 5V DC power to the servos and use the servo library on the ESP32 to write a specific angle. Repeat the proves with a higher voltage up to 7.5V.	Pass if the servos rotate at different speeds with the different voltages.	Behaves as expected.	P

Table 36: Individual Components' Unit Tests

7. References

No references were used for this document.

Feedback Integration

TA Feedback	Integration Comments
Include lower-level unit tests, such as testing software and hardware in isolation. Test sections of code.	Sections of code have been tested by observing values outputted through print statements. For instance, the IR sensor is confirmed to work through observing the value of IR fluctuate when exposed to an IR signal. Another example is observing the motor speed control function change the required speed over time.

Table 37: Feedback Integration

Appendix: Variables and Constants

A1. Monitored Variables

Name	Unit	Description
m_plant_id	-	The plant's identification details in order to control how much water is supplied at specific times.
m_plant_water_level	mL	Measures the moisture level of the plant. Each plant will have its own moisture sensor and value.
m_reservoir_level	mL	Measures the volume of water in the robots on board reservoir.
m_plant_location	Vector in m <x, y, z>	Determines the location of the plant relative to the robot.
m_ir_distance	m	Distance between the IR Emitter at the destination and IR Receivers on the robot
m_water_valve_position	-	Tracks whether the valve for watering mechanism is open or closed.
m_soil_distance	m	Determines if the nozzle of the water mechanism is above soil or not.
m_watering_arm_location	Vector in m <x, y, z>	Location of end effector of robot watering arm, relative to the robot.
m_robot_location	Vector in m <x, y, z>	Location of the robot relative to its environment, such as its initial starting position and the plant.
m_robot_orientation	Vector in degrees <α, β, ϴ>	The orientation of the robot relative to its starting position.
m_obstacle_distance	m	Location of an obstacle relative to the robot.
m_color_sensor_output	Vector in RGB	The RGB output of the color sensor

Table 38: Monitored Variables

A2. Controlled Variables

Name	Unit	Description
c_water_valve_position	-	Tracks whether the valve for watering mechanism is open or closed.
c_watering_arm_location	Vector in m <x, y, z>	Location of end effector of robot watering arm, relative to the robot.
c_water_pump_flow	mL/s	Speed of the water being pumped.
c_robot_speed	m/s	Overall speed of the robot when it is moving.
c_robot_location	Vector in m <x, y, z>	Location of the robot relative to its environment, such as its initial starting position and the plant.
c_robot_orientation	Vector in degrees <α, β, ϴ>	The orientation of the robot relative to its starting position.
c_plant_water_needed	Bool	Tracks whether a specific plant needs to be watered or not.

Table 39: Controlled Variables

A3. System Constants

Name	Unit	Description
NOMINAL_TEMPERATURE	°C	The expected room temperature
NOMINAL_PRESSURE	Atm	The expected air pressure in the room
BATTERY_VOLTAGE	V	The maximum voltage available to supply robot components
MOTOR_TORQUE	Nm	The torque of all motors used for movement will be constant.
POT_HEIGHT_RANGE	m	The range of heights that is allowable for a pot.
RESERVOIR_CAPACITY	L	The maximum water the on bord reservoir can hold.
RESERVOIR_CAPACITY_LOW	L	The threshold for when the water in the reservoir is considered low, for the user to be notified.
MIN_DISTANCE	m	The minimum distance that the robot must leave from all environmental obstacles.
MAX_FORCE	N	The maximum external force that the robot must receive before stopping operations.
MAX_SPEED	m/s	The maximum speed the robot should be allowed to move at.
MAX_ACCELERATION	m/s^2	The maximum acceleration that the robot should be allowed to move at.
BASE_LOCATION	Vector in metre <x, y, z>	The position where the robot stays when it is not required to water plants.
MAX_ATTEMPT_TIME	s	The maximum time the robot can attempt to reach a plant before it must stop.
MAX_RESPONSE_TIME	s	The maximum duration allowable for the user and system's interaction, in either direction.
MAX_RADIUS	m	The maximum radius within which the robot can detect and identify plants to water.
RESERVOIR_CAPACITY_LOW	%	The percentage water reservoir level that the user will be notified at, once it is reached.
BATTERY_LEVEL_LOW	%	The percentage battery level that the user will be notified at, once it is reached.

Table 40: System Constants