MEMORANDUM

To:	Charlie Refvem, Department of Mechanical Engineering, Cal Poly SLO	
	crefvem@calpoly.edu	
From:	Antonio Ventimiglia	Caiden Bonney
	ventimig@calpoly.edu	clbonney@calpoly.edu
Date:	10/23/2025	
RE:	**Lab 0x02: Tasks, Shares, and the Scheduler**	

As development of the Romi board progressed, the team began developing the robot's general code architecture. Each of Romi's actions was woven into a scheduler that "multitasks" cooperatively, allowing each task to be executed at a specified period (or at some time close to that).

Romi's behavior is structured around five primary tasks: User Input, Left Motor Controller, Right Motor Controller, Data Transfer, and Garbage Collection. Each task is assigned a priority and execution period. Tasks are scheduled in descending priority order. The highest-priority task always preempts lower-priority ones. A task with lower priority will only be executed if all higher-priority tasks are idling. The period defines the interval at which each task attempts to execute. However, task execution is not strictly periodic. Higher-priority tasks may introduce latency, causing lower-priority tasks to miss their intended execution time.

Inter-task communication is achieved through shared variables, implemented as either Shares or Queues. A Share is a single-value container, typically used for Boolean flags or static input parameters. Its simplicity makes it suitable for conveying discrete states or conditions. In contrast, a Queue is designed to store multiple values in a sequential array and is primarily used for dynamic data exchange. Data is enqueued and dequeued in the order of arrival (FIFO), thereby preserving temporal integrity during transmission. This communication mechanism supports efficient data flow between tasks while protecting crucial data from detrimental corruption. All inter-task communication variables are shown in Table 1.

Table 1. All Shared Variables and Their Purposes

Name	Data Type	Purpose
l_flag_s	Share: uint8	Input is changing left motor
r_flag_s	Share: uint8	Input is changing right motor
l_speed_s	Share: float	The speed set for the left motor
r_speed_s	Share: float	The speed set for the right motor
data_transfer_s	Share: uint8	Data should be communicated externally
test_complete_s	Share: uint8	Test actions and comms have been completed
l_time_q	Queue: uint32	Left encoder's time data
r_time_q	Queue: uint32	Right encoder's time data
l_pos_q	Queue: int32	Left encoder's position data
r_pos_q	Queue: int32	Right encoder's position data
l_vel_q	Queue: int32	Left encoder's velocity data
r_vel_q	Queue: int32	Right encoder's velocity data

The User Input task manages all incoming commands to the Romi, whether received via USB or Bluetooth. These commands are distributed to other tasks through the following shares: “l_flag_s”, “r_flag_s,” “l_speed_s,” “r_speed_s,” “data_transfer_s,” and “test_complete_s.” The data type and function of each variable are summarized in Table 1. User Input enables four primary actions: selecting which motor to control, setting the speed of either motor, determining whether to transmit data externally, and indicating when a test has been manually ended. These actions were determined to be all-encompassing of the functionalities the operator would ask of Romi.

The Motor Controller task is identical for both the left and right motors. This task is responsible for setting the motor speeds as communicated from User Input. Additionally, it is responsible for encoder handling. The position and velocity data from the encoders are time-stamped and communicated to Data Transfer via the queues “l_time_q,” “l_pos_q,” “l_vel_q,” “r_time_q,” “r_pos_q,” and “r_vel_q.” Again, these variables are expanded on in Table 1. Motor Controller also sets the “test_complete_s” share when the predetermined test time limit has elapsed. This signals to Data Transfer that data is no longer being fed into the queues.

The Data Transfer task is responsible for communicating all tracked data from the Romi to an external device, whether via USB or Bluetooth. It receives this data from the Motor Controller tasks through the queues mentioned above. It also receives the “test_complete_s” share, which allows Data Transfer to stop checking for data once the last data set is sent.

Lastly, the Garbage Collector task’s sole responsibility is to run the garbage collection function, which defragments memory, while the other tasks are not performing any work. Figure 1 illustrates the information above for each task, including its period and priority.

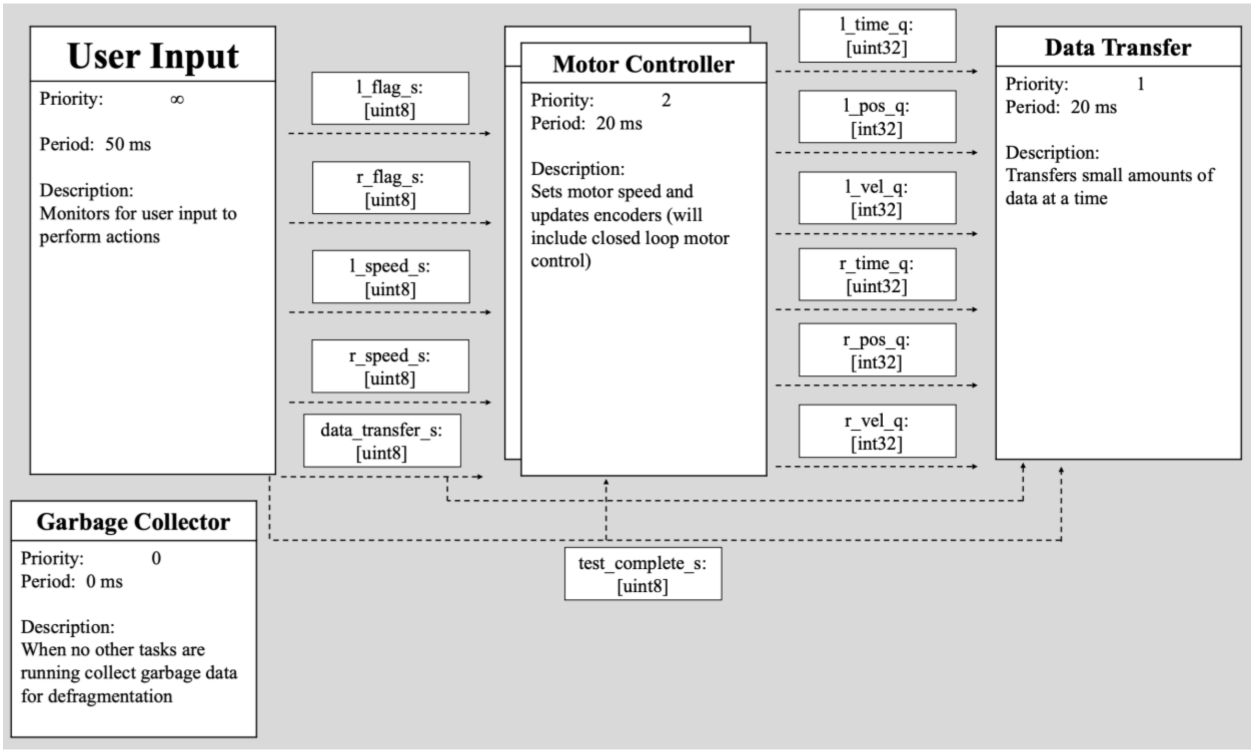
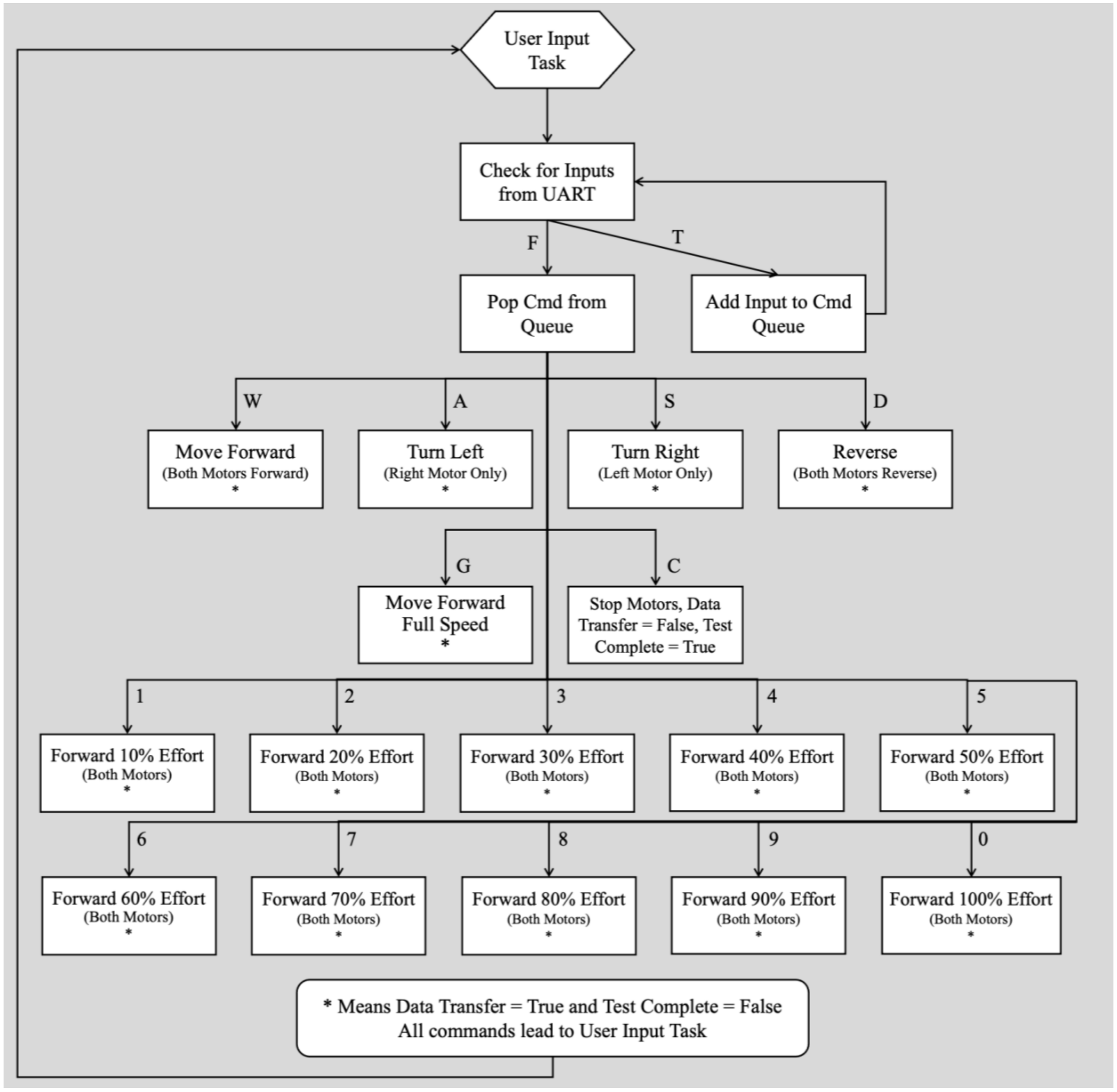


Figure 1. Task Diagram for Romi

Figures 2 through 5 illustrate the functionality of each task. Since a finite space machine (FSM) architecture was not chosen for our code, the following figures illustrate the tasks according to their "Boolean trees."



User Input Task
Check for Inputs from UART
F
T
Pop Cmd from Queue
Add Input to Cmd Queue
W
A
S
D
Move Forward (Both Motors Forward) *
Turn Left (Right Motor Only) *
Turn Right (Left Motor Only) *
Reverse (Both Motors Reverse) *
G
C
Move Forward Full Speed *
Stop Motors, Data Transfer = False, Test Complete = True
1
2
3
4
5
Forward 10% Effort (Both Motors) *
Forward 20% Effort (Both Motors) *
Forward 30% Effort (Both Motors) *
Forward 40% Effort (Both Motors) *
Forward 50% Effort (Both Motors) *
6
7
8
9
0
Forward 60% Effort (Both Motors) *
Forward 70% Effort (Both Motors) *
Forward 80% Effort (Both Motors) *
Forward 90% Effort (Both Motors) *
Forward 100% Effort (Both Motors) *
* Means Data Transfer = True and Test Complete = False
All commands lead to User Input Task

Figure 2. User Input Task Specific Diagram

The User Input Task receives user input through the Bluetooth UART port and performs the given command if one exists otherwise the command is ignored.

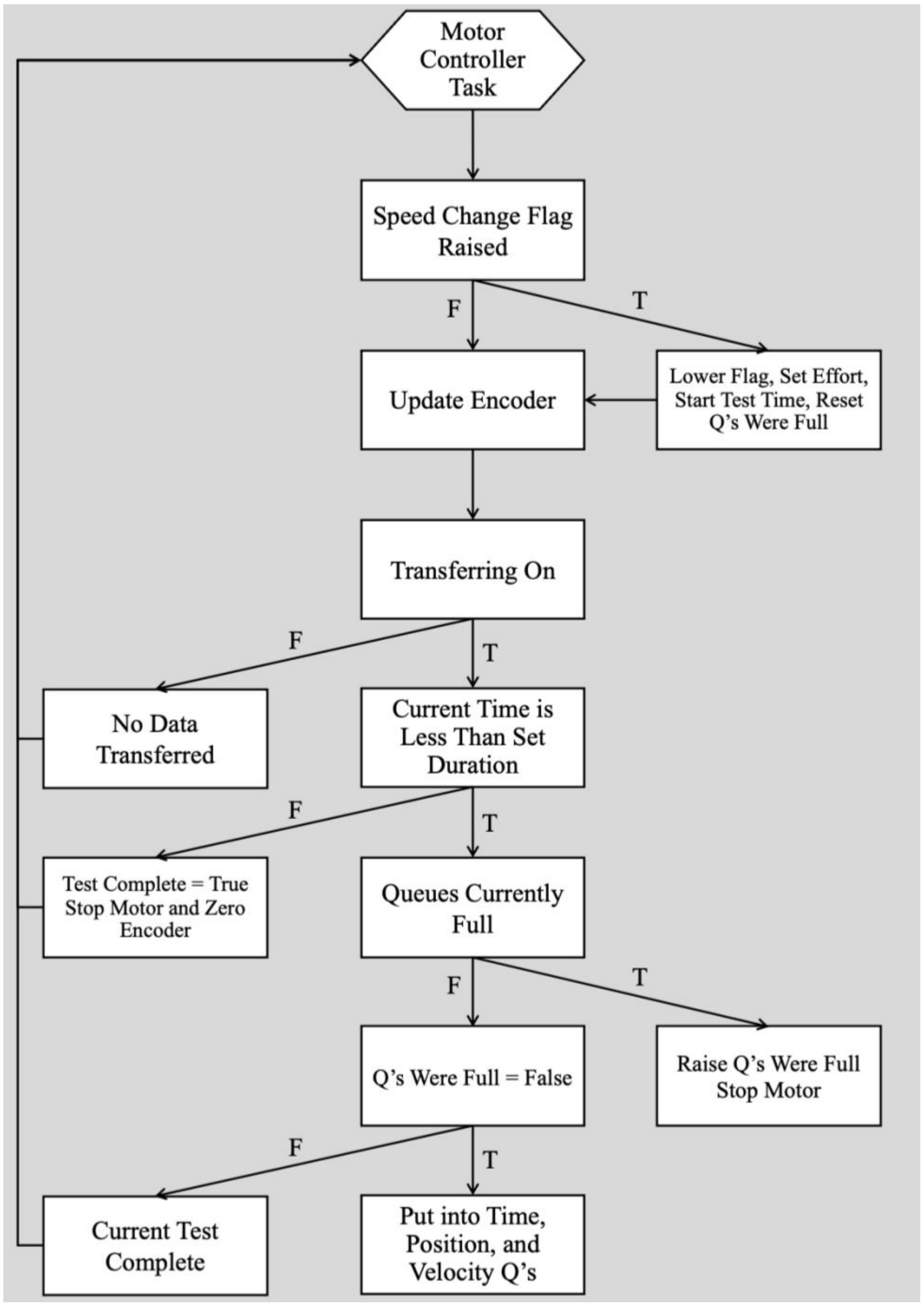

Motor Controller Task
Speed Change Flag Raised
F
T
Update Encoder
Lower Flag, Set Effort, Start Test Time, Reset Q's Were Full
Transferring On
F
T
No Data Transferred
Current Time is Less Than Set Duration
F
T
Test Complete = True Stop Motor and Zero Encoder
Queues Currently Full
F
T
Q's Were Full = False
Raise Q's Were Full Stop Motor
F
T
Current Test Complete
Put into Time, Position, and Velocity Q's

Figure 3. Motor Controller Task Specific Diagram

The Motor Controller Task overall links the motor to is respective motor in a single task such that the speeds of the motor can be changed based on the values read in from the encoder. This will be implemented in the following lab in the form of a control loop. The current Motor Controller Task updates the speed of the motor based on the left/right speed flags raised by the User Input Task while simultaneously updating the encoders. Data from the encoders last update is then placed into queues for the Data Transfer Task to access. Motor Controller only places the data into the queues if the data transfer flag is raised, current test has surpassed its allotted 2 second duration, or the queues have filled completely.

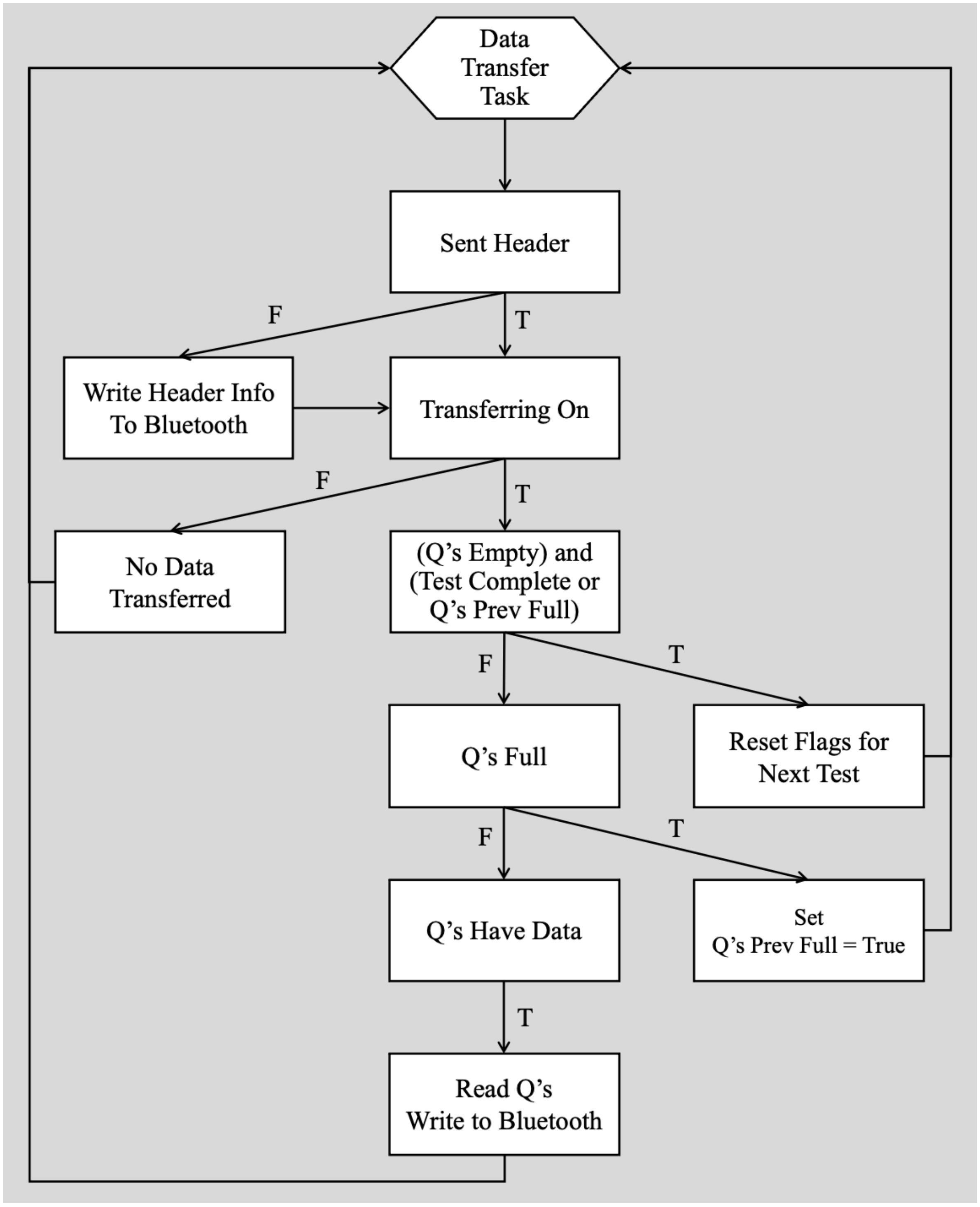

Data Transfer Task
Sent Header
F
T
Write Header Info To Bluetooth
Transferring On
F
T
No Data Transferred
(Q's Empty) and (Test Complete or Q's Prev Full)
F
T
Q's Full
Reset Flags for Next Test
F
T
Q's Have Data
Set Q's Prev Full = True
T
Read Q's Write to Bluetooth

Figure 4. Data Transfer Task Specific Diagram

The Data Transfer Task transfers data from the Romi across Bluetooth through the use of the Romi's UART port to the host of the Bluetooth connection. If this is the first transfer of the test then the headers are transmitted ensuring each tests data is clearly separated. A data transfer flag determines whether the data should be transmitted across Bluetooth or not. If the data transfer flag is raised and the queues contain data then one data point from all queues are transmitted simultaneously.

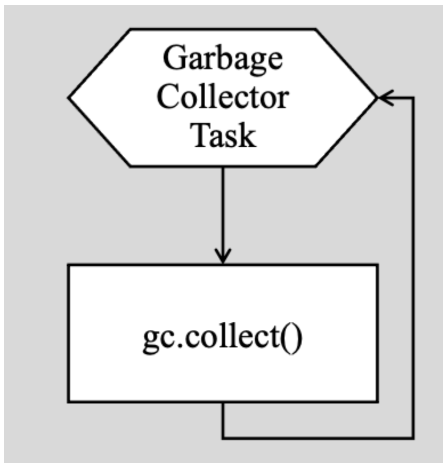


Figure 5. Garbage Collector Task Specific Diagram

The Garbage Collector Task runs in the free time between the higher priority tasks previously noted. The only purpose of the task is to defragment as much of the memory as possible to ensure as much of the data is stored in memory efficiently.

As seen in Figure 1, the priority order of the tasks (descending) is: User Input, Motor Controller, Data Transfer, and Garbage Collector. User Input has the highest priority, since the data sent to Romi has the highest urgency. If an operator is trying to end a task due to an emergency or something breaking, the command must be read and not blocked by other functionalities. The period was chosen to be 50 ms because moderate operator "input mashing" was observed to happen at this time scale. Additionally, having an unnecessarily low period would block the other tasks from executing. The Motor Controller task has the second-highest priority since it houses the encoder reading functionality. The encoders need to be read at an appropriate interval such that the velocity data is accurate and the auto-reload of the encoder register can be adequately detected. If this task runs too slowly, the data will be nonsense. It was determined that a reading frequency of 50Hz is appropriate to address the issues raised, which corresponds to the 20ms period shown in Figure 1. Data Transfer is the next highest priority. It was decided that if the data wasn't being transferred because the Motor Controllers were too busy, this sacrifice was acceptable. The period chosen for Data Transfer was the smallest time interval that allowed the task to function consistently. Periods of lower values would cause the data to be corrupted, which is theorized to be a hardware constraint imposed by the cheap Bluetooth module being used on the Romi. Lastly, the Garbage Collector. Its functionality had the lowest priority since it is only done for convenience. The period was set to zero, because it was designed to run when there is time between the other tasks.

To ensure that the tasks were all running at the frequency we had set them, the Romi performed multiple tests, with each task tracked by the profiler in the code architecture. After execution, the profiler provided the runtime information found in Table 2.

Table 2. Profiler Output for Each Task

	PRI	PERIOD	RUNS	AVG DUR	MAX DUR	AVG LATE	MAX LATE
User Input Task	10	50	2096	0.131	3.386	2.407	132.312
Left Motor Controller Task	2	20	5241	0.437	90.207	2.448	141.34
Right Motor Controller Task	2	20	5241	0.41	1.237	2.598	159.415
Data Transfer Task	1	20	5241	0.284	4.775	3.143	293.191
Garbage Collection Task	0	0	20302	3.492	138.52	52620	104836

As shown in Table 2, most tasks are running a little late on average. This little bit of latency does not affect the functionality of the tasks. Garbage Collection is the exception. It is acceptable for Garbage Collection not to run for extended periods, since it was discussed that it is a nicety in the code. Overall, the priorities and periods are causing the code to behave as intended.

Testing was conducted using a script on a local machine that automatically interfaced with the Romi. The operator simply entered a value from 0 to 9, where each number corresponded to a specific motor effort level: 1 = 10%, 2 = 20%, … 9 = 90%, and 0 = 100%. For each effort level, the position and velocity data of both wheels were recorded and saved to CSV files. Examples of these CSV outputs are provided in Appendix A. The collected data were automatically processed and plotted, with the resulting position and velocity plots for each wheel presented in Appendix B, along with the corresponding data analysis. Appendix C lists the files used to run the Romi tests, along with a brief description of their functionality.

Appendix A: Example CSV Outputs from Romi Tests

A.1 Part of 40% Effort Data Log

```
lt,rt,lp,rp,lv,rv
459.0,449.0,94.0,79.0,0.0,0.0
20298.0,20115.0,103.0,90.0,453.0,559.0
40765.0,40578.0,123.0,114.0,977.0,1172.0
61804.0,61622.0,148.0,143.0,1188.0,1378.0
78101.0,77915.0,170.0,168.0,1349.0,1534.0
99254.0,99071.0,202.0,204.0,1513.0,1701.0
119982.0,119796.0,237.0,243.0,1688.0,1881.0
140872.0,140688.0,275.0,287.0,1819.0,2106.0
157753.0,157566.0,308.0,324.0,1954.0,2192.0
178644.0,178460.0,351.0,371.0,2058.0,2249.0
199990.0,199804.0,395.0,421.0,2061.0,2342.0
220882.0,220699.0,439.0,471.0,2105.0,2393.0
241787.0,241601.0,485.0,522.0,2200.0,2439.0
258656.0,258472.0,522.0,564.0,2193.0,2489.0
279563.0,279377.0,569.0,617.0,2248.0,2535.0
300895.0,300711.0,617.0,671.0,2250.0,2531.0
321802.0,321616.0,664.0,723.0,2247.0,2487.0
338258.0,338076.0,701.0,765.0,2247.0,2551.0
359604.0,359418.0,750.0,818.0,2296.0,2483.0
380499.0,380315.0,797.0,871.0,2249.0,2536.0
397411.0,397225.0,835.0,914.0,2246.0,2543.0
418298.0,418116.0,882.0,967.0,2250.0,2536.0
439203.0,439018.0,930.0,1021.0,2296.0,2583.0
460593.0,460409.0,979.0,1075.0,2291.0,2524.0
481586.0,481399.0,1027.0,1129.0,2286.0,2572.0
498657.0,498473.0,1066.0,1174.0,2284.0,2635.0
519734.0,519547.0,1114.0,1228.0,2277.0,2562.0
540799.0,540615.0,1162.0,1282.0,2278.0,2563.0
557854.0,557667.0,1201.0,1327.0,2286.0,2638.0
578917.0,578733.0,1249.0,1381.0,2279.0,2563.0
600437.0,600249.0,1298.0,1437.0,2277.0,2602.0
621499.0,621315.0,1346.0,1491.0,2278.0,2563.0
638142.0,637957.0,1385.0,1534.0,2342.0,2584.0
659648.0,659465.0,1433.0,1590.0,2232.0,2603.0
680726.0,680539.0,1482.0,1644.0,2324.0,2562.0
697800.0,697616.0,1521.0,1688.0,2283.0,2576.0
718880.0,718692.0,1569.0,1742.0,2277.0,2562.0
739942.0,739760.0,1618.0,1797.0,2326.0,2610.0
761434.0,761247.0,1668.0,1853.0,2326.0,2606.0
778063.0,777881.0,1706.0,1896.0,2284.0,2585.0
```

A.2 Part of 80% Effort Data Log

```
lt,rt,lp,rp,lv,rv
456.0,449.0,252.0,211.0,0.0,0.0
20484.0,20287.0,271.0,229.0,949.0,907.0
37044.0,36843.0,303.0,262.0,1931.0,1992.0
57761.0,57563.0,356.0,317.0,2558.0,2654.0
78487.0,78286.0,418.0,383.0,2991.0,3185.0
99644.0,99446.0,489.0,459.0,3356.0,3591.0
120369.0,120168.0,564.0,538.0,3618.0,3812.0
137267.0,137068.0,629.0,606.0,3845.0,4023.0
158172.0,157970.0,711.0,693.0,3923.0,4162.0
179065.0,178866.0,795.0,782.0,4020.0,4259.0
200380.0,200181.0,885.0,875.0,4222.0,4363.0
216839.0,216641.0,956.0,946.0,4312.0,4313.0
238180.0,237978.0,1047.0,1037.0,4265.0,4264.0
259245.0,259046.0,1138.0,1129.0,4319.0,4366.0
280323.0,280121.0,1230.0,1220.0,4364.0,4317.0
297364.0,297167.0,1305.0,1293.0,4400.0,4283.0
318442.0,318241.0,1398.0,1385.0,4413.0,4365.0
339948.0,339749.0,1495.0,1481.0,4510.0,4463.0
361026.0,360824.0,1592.0,1578.0,4601.0,4602.0
377656.0,377459.0,1669.0,1655.0,4628.0,4628.0
399174.0,398972.0,1769.0,1754.0,4648.0,4602.0
420239.0,420040.0,1866.0,1852.0,4604.0,4651.0
437322.0,437120.0,1945.0,1931.0,4623.0,4625.0
458387.0,458190.0,2043.0,2029.0,4652.0,4651.0
479465.0,479266.0,2141.0,2127.0,4649.0,4650.0
500943.0,500744.0,2241.0,2227.0,4656.0,4655.0
517586.0,517385.0,2319.0,2305.0,4684.0,4687.0
539094.0,538896.0,2420.0,2406.0,4697.0,4695.0
560170.0,559968.0,2520.0,2505.0,4744.0,4697.0
576801.0,576602.0,2599.0,2583.0,4749.0,4689.0
598316.0,598115.0,2702.0,2684.0,4788.0,4695.0
619381.0,619185.0,2802.0,2783.0,4746.0,4698.0
640900.0,640698.0,2906.0,2885.0,4833.0,4741.0
657530.0,657333.0,2985.0,2963.0,4748.0,4689.0
678606.0,678407.0,3086.0,3062.0,4793.0,4697.0
700084.0,699886.0,3189.0,3164.0,4796.0,4748.0
716726.0,716526.0,3268.0,3241.0,4745.0,4627.0
738232.0,738034.0,3370.0,3342.0,4744.0,4695.0
759309.0,759108.0,3469.0,3441.0,4696.0,4697.0
780378.0,780181.0,3568.0,3540.0,4699.0,4698.0
797431.0,797230.0,3648.0,3620.0,4690.0,4692.0
```

Appendix B: Plot of Romi Tests With Analysis

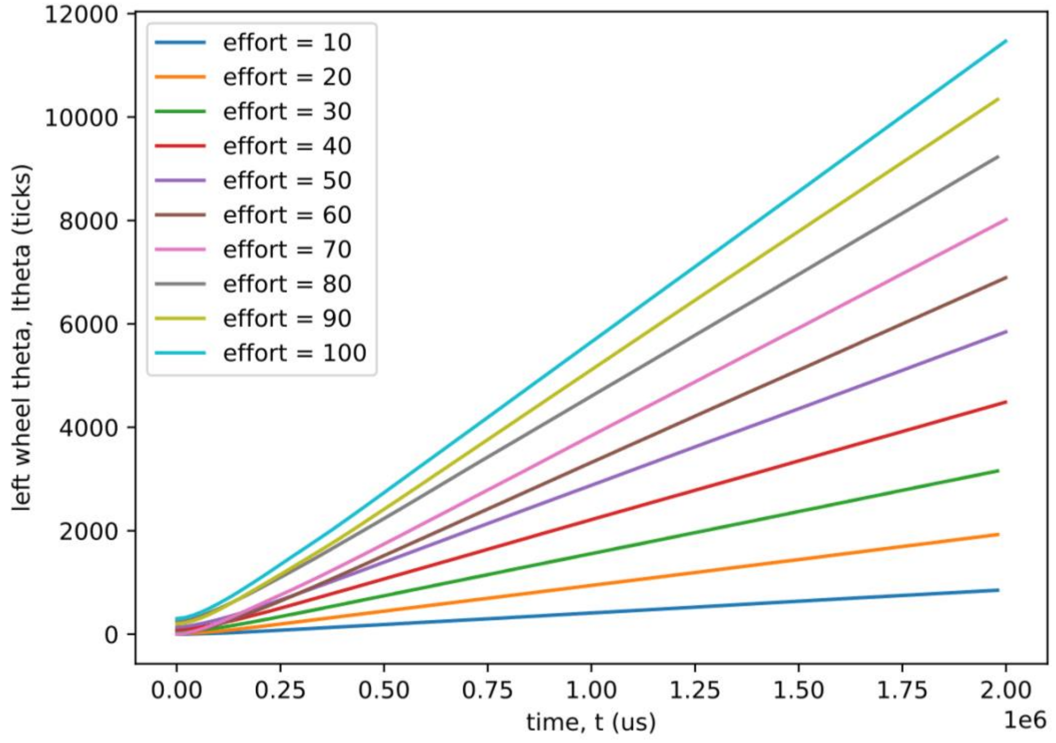


Figure B1. Left Theta vs Time

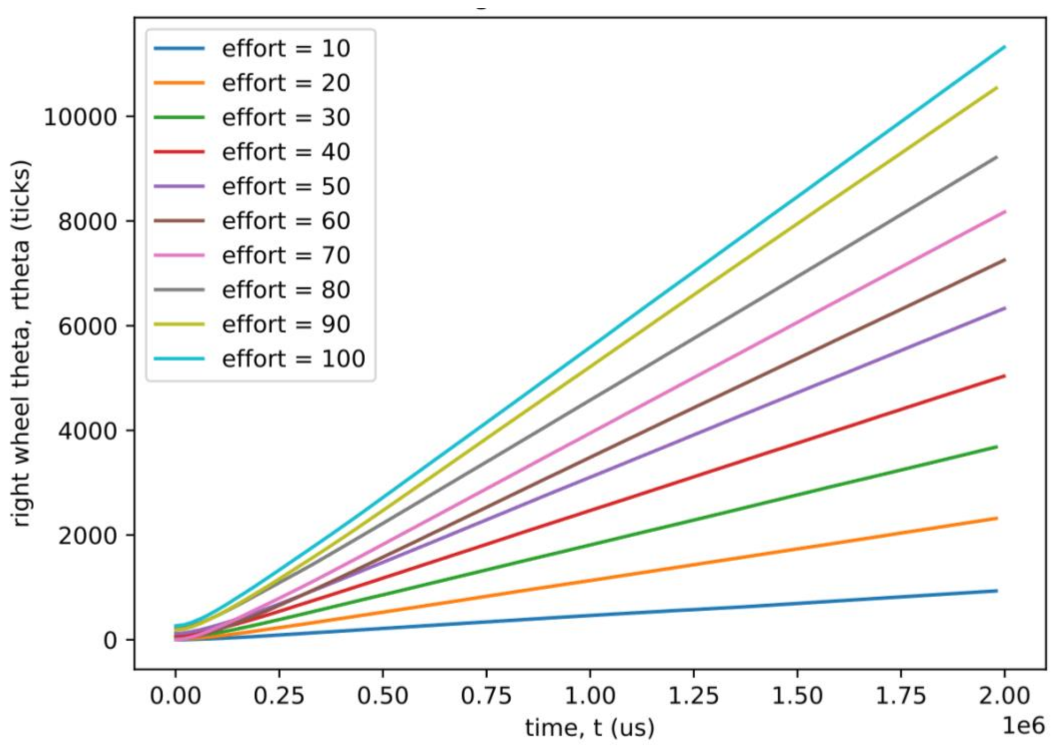


Figure B2. Right Theta vs Tim e

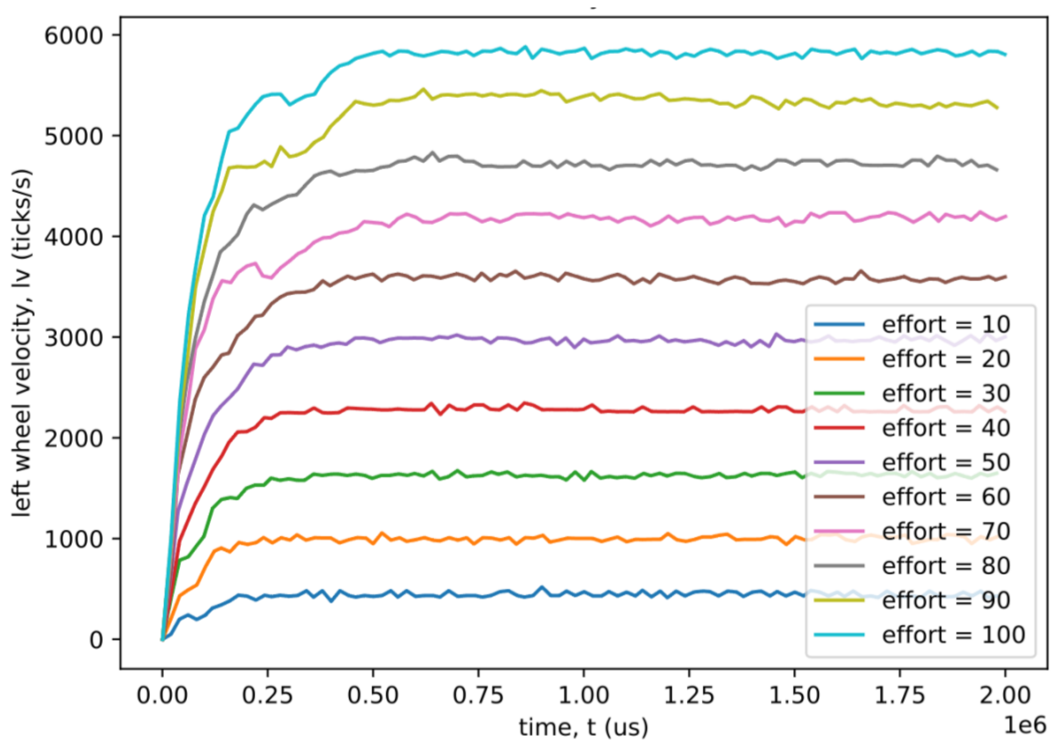


Figure B3. Left Velocity vs Time

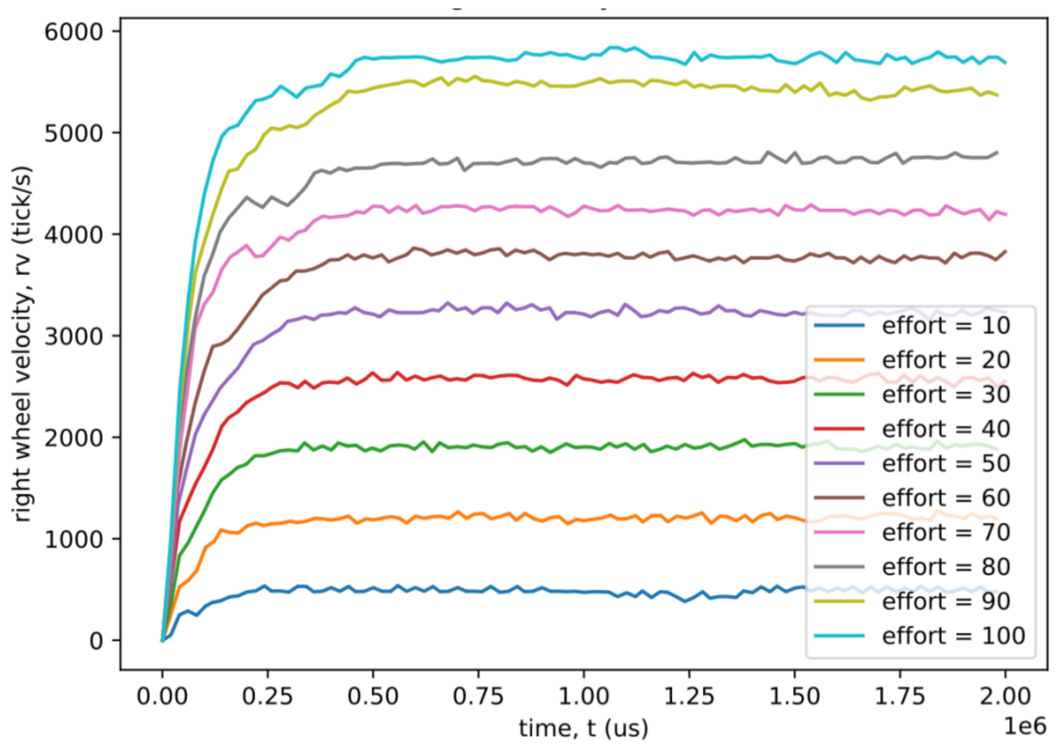


Figure B4. Right Velocity vs Time

Analysis of Testing

The Romi robot was tested by recording motor response data over 2-second intervals at effort levels ranging from 10% to 100% in the forward direction. Before each trial, the encoders were zeroed to ensure that the initial angular position and velocity were both zero. Each test consisted of a step input, where the motors were commanded from rest to the desired effort level. Although additional tests were conducted with the robot suspended in the air to observe frictional and inertial effects, the primary focus of this analysis is the grounded data. As shown in Figures B3 and B4, the higher-effort tests exhibit more noticeable irregularities in the velocity profiles. This behavior arises from the wheels intermittently gaining and losing traction on the moderately frictional lab surface, which causes brief changes in rotational speed. Since velocity is derived from the change in wheel angle over discrete time intervals, these traction effects appear as small variations in the recorded data. These results allow us to derive key system parameters that will inform motor control tuning and help achieve the desired motion responses.

Appendix C: Experiment Python Test Files

<u>Romi Files0</u>

1. main.py
    - Sets up motor timers, motors, and encoder objects with their correct pins on the Nucleo and initializes the shared variables that allow the cooperative tasks to communicate. Instantiates user input, motor control, data transfer, and garbage collection task objects and registers them with the cotask scheduler. Runs the priority scheduler in a loop, and on any exception stops the motors, clears data queues, prints task diagnostics, and re-raises the error.
2. cotask.py
    - Provides MicroPython cooperative multitasking support. Task wraps user generator functions with timing, profiling, and tracing, while TaskList manages tasks grouped by priority. The scheduler implements round-robin and priority-based execution, cycling tasks and respecting readiness. Includes diagnostic reporting for run counts, durations, and lateness, and exports a global task_list to be used by applications.
3. task_share.py
    - Implements lockable data structures for cooperative MicroPython tasks. Queue buffers typed items with optional overwrite, thread protection, diagnostic tracking, and helpers like put, get, full, clear, and any. Share provides a single-value exchange with interrupt-safe access, and a global registry plus show_all() render readable state summaries.
4. User_Input.py
    - Initializes a UART connection, buffers incoming single-character commands into a queue. The run coroutine pulls the user input from the queue, echoes commands, and maps WASD/g/c/0-9 keys to specific motor flag and speed updates shared with other tasks. It also toggles data-transfer and test-complete signals. If c is pressed it also sends a UART confirmation that the current test was completed for cancelation of a test currently running.
5. Motor_Controller,py
    - Creates a motor controller that enables a motor/encoder pair. In its run coroutine, it reacts to a flag to apply a new effort command, then streams timestamped position and velocity samples into queues while a timed test is active. If queues overflow or the duration elapses it stops the motor, marks the test complete, and rezeros the encoder.
6. Data_Transfer,py
    - Manages UART-based streaming of queued data to a host. After sending a CSV header once, the run coroutine waits for the data-transfer flag, then continuously drains synchronized samples from left/right time, position, and velocity queues, in CSV formatted rows. It detects queue exhaustion or completed tests to reset state, resends the header on the next run, and notifies the host that data transfer finished.
7. Garbage_Collector,py
    - Provides a cooperative task that continuously runs MicroPython's garbage collector. The run coroutine simply loops, calling gc.collect() to keep memory defragmented whenever the scheduler gives it time.

8. encoder.py
   - Utilizes a hardware timer configured for counting encoder ticks to track encoder position and velocity. The update function handles overflow/underflow while accumulating position, and computing the elapsed time between each update() call. It has a zero() method to reset the state of its two tracked properties, position and velocity.
9. motor.py
   - Configures a motor driver and contains simple control helpers for setting effort and enabling or disabling the motor. The constructor creates a Pin object for the given pins locations for PWM, DIR, and nSLP as well as assigning the specified channel on a given timer to the PWM pin. The set_effort() function maps -100 to 100 effort commands to their corresponding directions and PWM duty, while enable()/disable() toggle the driver's sleep state and reset the motors current effort.

<u>Computer Files</u>

10. Interface_w_Romi.py
    - Acts as a host to run tests on the Romi via Bluetooth serial, defaulting to /dev/cu.mecha16 unless a port is passed. It opens the connection, flushes the buffer, prompts the user for test IDs, sends them to the robot, and logs streamed CSV data until a "Test Data Transfer Complete" message arrives. Each run writes validated numeric rows to Lab 0x02/Results/log{test}.csv, reporting any malformed data and closing open files and serial ports on a test ID input of q. Also accepts keyboard interruptions in case of runaway tests.
11. CSV_Reader.py
    - Parses each CSV log, rejecting malformed lines, and extracts two selected columns to plot against each other. multifile_process_data() batches logs for efforts 10–100 and overlays their curves on shared axes. The main function generates and saves SVG plots for both wheels' position and velocity over time in Lab 0x02/Results.

References:

- Refvem, C. (2025). *Lab 0x02 Tasks, Shares, and the Scheduler* [Unpublished lab instructions]. Department of Mechanical Engineering, Cal Poly. Retrieved from https://canvas.calpoly.edu/courses/161863/assignments/1368602?module_item_id=4790191
- Refvem, C. (2025). *Lab 0x02 - Scheduler* [Unpublished lecture notes]. Department of Mechanical Engineering, Cal Poly.
- Refvem, C. (2025). *Lecture 5 - Decoding Quadrature Encoders* [Unpublished lecture notes]. Department of Mechanical Engineering, Cal Poly.
- Refvem, C. (2025). *Lecture 7 - Software Timing, Generators* [Unpublished lecture notes]. Department of Mechanical Engineering, Cal Poly.
- Refvem, C. (2025). *Lecture 8 - Task Diagrams and the Scheduler* [Unpublished lecture notes]. Department of Mechanical Engineering, Cal Poly.
- Refvem, C. (2025). *Lecture 9 – FSMs and State Transition Diagrams* [Unpublished lecture notes]. Department of Mechanical Engineering, Cal Poly.
- ChatGPT. (2025). *Coding Concepts and Debugging Consultation* [Unpublished AI-generated text]. OpenAI. Retrieved from https://chat.openai.com/