CPRE 288 Embedded Systems Platform Movement and Code Optimization

Final Design Document

Team SDMAY 20-04

Clients - Dr. Phillip Jones, Dr. Diane Rover
Advisor - Matthew Post

Team Members

Jacob Aspinall – Team communication manager

Geonhee Cho – Report manager

Jisoo Han – Team Git master

Sam Rai – Team website manager

Nathan Nordling – Team data manager

Issac Klein – Team Scribe

Team Email: sdmay20-04@iastate.edu

Team-website: **http://sdmay20-04.sd.ece.iastate.edu**

Version 4.0 (4/26/2020)

Executive Summary

Engineering Standards & Design Practices

- Agile development
- Unit testing
- Continuous integration

Summary of Requirements

- Automated unit tests for functionality of each device
  - Simulation of all external hardware(sensors, roomba, LCD etc.)
- Run tests with Gitlabs CI tools
- New organized git repositories
  - One repository for professors and TAs
  - One repository for students

Applicable Courses from Iowa State University Curriculum

- CPRE 288

New Skills/Knowledge acquired that was not taught in courses

- Continuous Integration
- Simulation
- Scripting
- Git

Table of Contents

1. Introduction 5

Acknowledgement 5

Problem and Project Statement 5

Operational Environment 5

Requirements 6

Intended Users and Uses 6

Assumptions and Limitations 6

1.6.1 Assumptions 6

1.6.2 Limitations 6

Expected End Product and Deliverables 6

2. Specifications and Analysis 8

Proposed Design 8

2.1.1 Unit testing 8

2.1.2 Continuous Integration 8

2.1.3 Simulation of external hardware 8

2.1.4 Git Repository 8

Design Analysis 8

2.2.1 Unit testing for all labs 8

2.2.2 Continuous Integration 9

2.2.3 External Hardware simulation 9

2.2.4 Git Repository 9

Development Process 9

Design Plan 10

3. Statement of Work 11

3.1 Previous Work And Literature 11

3.2 Technology Considerations 11

3.2.1 Software available for development on TI's TM4c123GH6pm microcontroller 11

3.2.2 Gitlab 11

3.3 Task Decomposition **Error! Bookmark not defined.**

3.3.1 Unit testing **Error! Bookmark not defined.**

3.3.2 Integration testing/automated testing **Error! Bookmark not defined.**

3.3.3 Simulation of external hardware 11

3.3.4 Git repository 12

3.3.5 Lab demo videos 12

3.4 Possible Risks And Risk Management 12

3.5 Project Proposed Milestones and Evaluation Criteria 12

3.5.1 Unit testing milestones 12

3.5.2 Integration Testing/Automated Testing 12

3.5.3 Simulation of the external hardware 13

3.5.4 Git repository 13

3.6 Project Tracking Procedures 13

3.7 Expected Results and Validation 13

4. Project Timeline, Estimated Resources, and Challenges 15

4.1 Project Timeline 15

4.2 Feasibility Assessment 15

4.3 Personnel Effort Requirements 16

4.4 Other Resource Requirements 19

5. Testing and Implementation 19

Interface Specifications 19

Hardware and software 19

5.2.1 Code Composer Studio(CSS) 19

5.2.2 Debug Server Scripting(DSS) 20

5.2.3 Simulation of external hardware 20

5.2.4 GitLab built in Continuous Integration tool 20

Functional Testing 20

5.3.1 Unit Testing 20

5.3.2 Integration Testing 21

5.3.3 System Level testing 21

5.3.4 Manual Testing 21

6. Closing Material 22

6.1 Conclusion 22

6.2 References 22

List of figures/tables/symbols/definitions

Figure 2.1: High Level Project Diagram 10

Figure 4.1: Project Timeline 15

Figure 5.1: Testing Process 22

1. Introduction

CPRE 288 is the introductory class to embedded systems at Iowa State University. Students use the Cybot platform, which is a combination of a Roomba, sonar and infrared distance sensors mounted on a servo, and a Wi-Fi access point. All controlled from Texas Instrument's TM4c123gh6pm microcontroller. The laboratory assignments for the class are to let the students progressively learn and build software libraries using the C programming language for each component, with the goal to combine them all into one final project by the end of the semester.

1.1 Acknowledgement

Isaac Rex - for initial work and help on the Roombas movement functionality, Open Interface driver code, and the Timer. C file

1.2 Problem and Project Statement

The current git repository for CPRE 288, the intro to embedded systems class at Iowa State University, holds the solutions for each weekly lab assignment for the class. However, it has not been reviewed thoroughly since the labs were created. There are several known issues with the code, and potentially more that need to be fixed. There have also been updates and additions to the labs, and these need to be added to the codebase. The current Git repo is also unorganized, and there may be unnecessary files that need to be removed.

Our proposed solution is to first do a thorough review of the code, and to make sure each lab is updated to reflect the current requirements, then develop a series of automated unit tests, and connect them to a git repository for continuous integration. Because many labs require input or output to and from the outside world, we will need to build a simulation of the external hardware (i.e. distance sensors, Roomba). This enables us to test different inputs and outputs automatically without manual human intervention. The result will be git repositories that will be helpful for teachers and students and will be able to handle further developments of the course.

1.3 Operational Environment

The operational environment of the CyBot platform is inside the labs in Coover hall. The floors of these labs are flat and not carpet. Because Roombas are designed for vacuuming carpet, the wheels are operating on the surface they were not specifically designed for.

The sonar and infrared sensors are used to detect white pvc pipes with radius of 3 to 9cm. These pipes are randomly located and could be close together or far apart.

The Codebase we create will be used by Professors and TAs of CPRE 288 to assist in teaching students. Some of the code may be precompiled before given to students and will not be able to be changed by students on the fly.

1.4 Requirements

- Unit tests for all Device libraries
- Unit tests run automatically with GitLabs CI tools
- Simulation of external hardware to facilitate creation of automated unit tests
- Reorganized git repositories for professors and students

1.5 Intended Users and Uses

The intended users are the professors, teaching assistants, students, and any other developers for CPRE 288.

professors, TAs, and other developers will use the git repository to store files related to the course, including but not limited to lab solutions, device libraries, lab manuals, and datasheets. The automated unit tests will be used to ensure correctness of the device libraries when changes are made.

Students will use their git repository to view files given to them to use in the course, including but not limited to lab manuals, lab template code, and demo videos for each lab.

1.6 Assumptions and Limitations

1.6.1 Assumptions

- The operating conditions of the CyBot will only be as it is used currently in CPRE 288

1.6.2 Limitations

- Simulated hardware devices may not act exactly the same as an actual hardware device
- Due to differences in hardware on every CyBot, code that is calibrated to a certain CyBot, in particular code for the IR sensor and servo, may not work as intended on other CyBots

1.7 Expected End Product and Deliverables

- Unit tests for all device libraries
    - All requirements for each device will have unit tests that verify correctness. These unit test will also be automated with Gitlab's CI tools

      devices to be tested:

        - Roomba Open Interface
        - Sonar(Ping) sensor
        - Infrared sensor
        - timer. C helper file
        - Servo
        - UART
        - LCD
        - Buttons
- Continuous Integration
    - The unit tests will be run automatically on a commit using Gitlab's CI tools.
- Simulation of CyBot's external hardware
    - In order to perform automated tests, the external hardware (distance sensors, Roomba, etc.) will be simulated. This will be done on a separate microcontroller to the one running the code being tested. These two microcontrollers will be wired together. The simulation will mock inputs and record outputs to and from the code being tested.
- Organized git repos
    - Git repository for Professors, TAs and other developers. This repository will hold lab solutions, device libraries, and any other material needed for the course but not used by students
    - Git repository for Students. This repository will hold files needed by students for the course, including lab manuals, lab template code, lab demo videos, and any other files students need access to for the course.

2. Specifications and Analysis

2.1 Proposed Design

2.1.1 Unit testing

For each device we will generate a list of requirements for its libraries and ensure that each required functionality is implemented. We will then develop a set of unit tests to test these requirements to ensure correctness.

2.1.2 Continuous Integration

We will connect our git repository to Gitlab's Continuous Integration tools to run the unit tests automatically. Our simulation platform will be connected by USB to a server running a Gitlab runner and will stream the results of the tests from the microcontroller back to the server, and then back to the Gitlab repository.

2.1.3 Simulation of external hardware

To allow for automated testing, we need a non-manual way of providing different inputs to our code. Because a large part of the code relies on sensor inputs or outputs to other devices, we need to simulate the functionality of all or some of the hardware. The preferred method from the standpoint of easy testing would be a pure software simulation of the entire system. However, after researching, the only simulators available just simulate the Cortex-M4 processor, and not the rest of the peripheral devices on the microcontroller. Since there are already multiple hardware issues with the microcontrollers that affect our lab code, it is best that we test using the actual microcontroller.

The external hardware still must be simulated. We plan to write simulation programs for each device and run them on a separate microcontroller. This will then be wired to the main microcontroller to simulate inputs and collect outputs to and from the code we are testing.

The simulation microcontroller will have an API, accessed over UART, where the microcontroller connected to it can change the state of the system(i.e. change the servo position) or query for the current state of some device(i.e. query the current servo position).

2.1.4 Git Repository

We will construct 2 new git repositories for the course, one for professors, TAs, and any other developers, and one for students or other normal users. These will include the relevant files needed for each group. Our automated unit tests will be connected to the professor/TA repo.

2.2 Design Analysis

2.2.1 Unit testing for all labs

Code Composer, the IDE we use to run code on our microcontroller comes with a program called DSS (Debug server scripting). DSS allows us to control the debug server connected with the microcontroller with JavaScript. This allows us to write a script which can build and load code onto the microcontroller automatically.

The debug server also provides a printf() function, which under normal operation sends a string over the debug USB and prints to the IDE console window. DSS can also be used to send this string to a different file, which is what we do when linking our unit tests with Gitlab's CI tools.

Our tests are all written as C code running on the microcontroller the results are represented as strings which are printed to the console using the debug server's printf() function.

2.2.2 Continuous Integration

We use GitLab's built in CI tools to run the unit tests automatically after a commit. Our simulation platform is connected to a server running a Gitlab runner. A script will run the tests on the microcontroller using DSS (see section above). The test results are written to a file and then parsed to determine if any tests have failed or not. The result is sent back to Gitlab.

There will be two stages in the Continuous Integration pipeline, one for building the code, and one for testing. This allows us to easily see where the code failed.

2.2.3 External Hardware simulation

To simulate the external hardware 2 microcontrollers are wired together, one of them represents the actual microcontroller running the code to be tested(main microcontroller), the other runs a program simulating all of the external devices(sensors, servo, etc.)(simulation microcontroller).

The idea is that the code on the main microcontroller will behave just as it would in the real word. The simulation will send signals exactly mimicking the real devices.

The simulation will also keep track of the current state of the outside world (i.e. current servo position, coordinates of Roomba, the LCD display).

An API is provided to the main microcontroller to query the current state of the outside world. This allows our tests to view the results of any code run.

2.2.4 Git Repository

The 2 repositories will be hosted on Iowa State's Gitlab server.

2.3 Development Process

We are following an Agile process to allow frequent feedback from our clients. Since our clients are professors of computer engineering, and they both teach the CPRE 288 class, they may have extremely valuable insights and feedback on our process. It is very important that we can show off our progress frequently so we can constantly refine our process.

We have weekly or biweekly meetings with our adviser and clients.

2.4 Design Plan

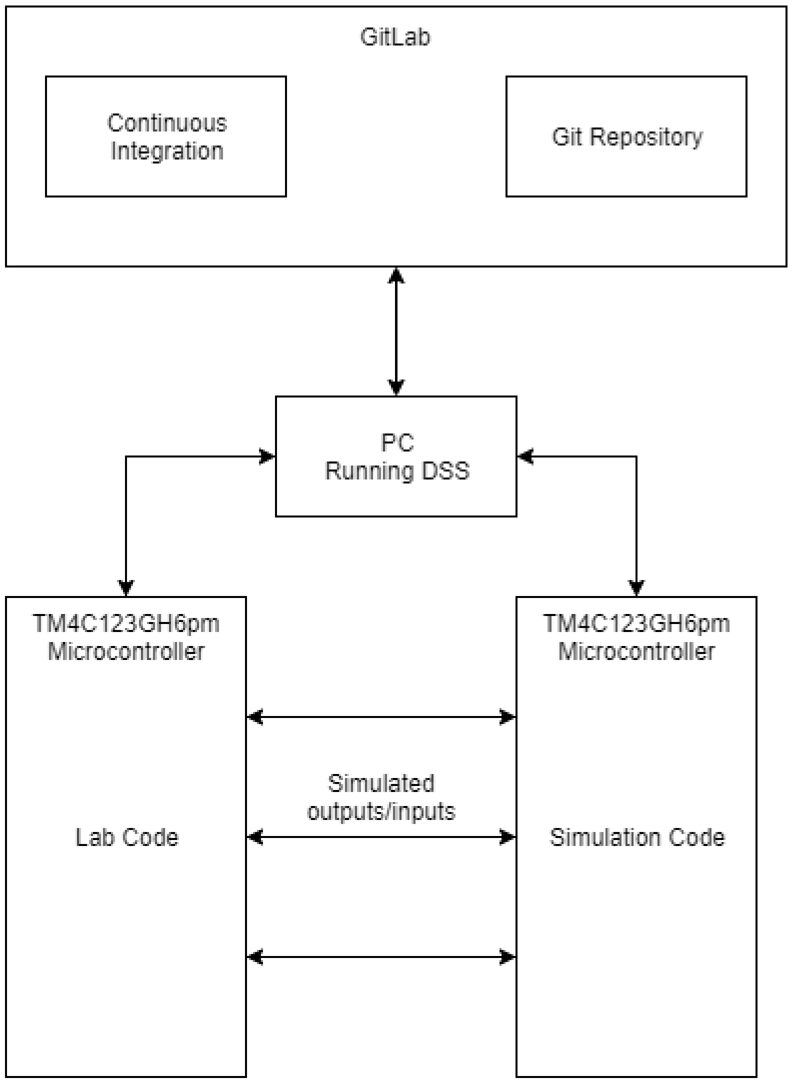


Figure 2.1: High

Our proposed design consists of three parts, our Gitlab server, our simulation platform, and the PC that the simulation platform is connected to. On a commit to the Git repo, Gitlab will notify the PC and will start the tests on the simulation platform. Our simulation platform is made up of two microcontrollers. One runs the simulations of external devices, and the other which mimics the actual CyBot microcontroller. The simulation platform runs the tests and reports back to the PC it is connected to. The PC will report the results back to Gitlab where the tests are indicated as either passed or failed.

3. Statement of Work

3.1 Previous Work And Literature

Our entire project is starting from the work done by the original writers of the current labs for CPRE 288. This work resides in a git repository currently owned by one of our clients Dr. Phillip Jones. Since the repository was first created, only minor updates have been made.

Starting from an already existing repository of complete labs puts us in a very advantageous position. Because it provides us with the solutions, we will only need to add updates and optimizations, not write the code from scratch. This allows us to focus more of our time on testing rather than development.

3.2 Technology Considerations

3.2.1 Software available for development on TI's TM4c123GH6pm microcontroller

TI provides a whole portfolio of software to help with software development on their chips. This includes a fully featured IDE (Code Composer), code libraries, and the ability to write scripts to control the debugger with DSS. These features make testing and development on the microcontrollers much more streamlined.

3.2.2 Gitlab

By using Gitlab to host our repository, we gain access to other tools. This includes built in support for continuous integration, an issues tracker/board, and a wiki that can be used for holding documentation. Gitlab allows us to have these project management and testing features all within one program, instead of having multiple tools.

1. Use DSS to be able to load code onto the microcontroller (Use build to run bash script)
2. Use DSS to be able to control the debugger
    a. Breakpoints, changing variable values; can print out the breakpoint result
3. Use GitLab's built in CI/CD platform to trigger a DSS script when changes are made to the git repository
4. Use DSS to assist in creating automated unit tests for each device

3.3.3 Simulation of external hardware

1. Simulation of each external device
    a. Ping sensor
    b. IR sensor
    c. Sending/receiving over UART
    d. Servo

    e. Buttons
    f. Roomba
    g. LCD

3.3.4 Git repository

1. Update projects to latest version of Code Compose/Tivaware libraries
2. Remove unneeded files/directories
3. Create 2 repositories, one for students and one of professors/TAs

3.3.5 Lab demo videos

1. Update each of the lab videos at Git repository.
2. Code review while filming demo videos.

3.4 Possible Risks And Risk Management

Currently the only area of concern is deep knowledge of programming for our specific microcontroller. Due to the fact that most of the labs rely on register level programming of the microcontrollers hardware, in our first semester, it is important to have each member who does not feel comfortable with register level programming to relearn and become an expert.

3.5 Project Proposed Milestones and Evaluation Criteria

3.5.1 Unit testing milestones

1. Sonar(PING) sensor
    a. The lab associated with the sonar sensor is generally the most problematic lab for students of 288. It currently has several questions that need to be answered regarding the proper implementation.
2. Lab 6(IR), Timer .c, open interface
    a. Lab 6 is another complex lab that might be able to be optimized. The Timer .c helper file and the open interface have already been looked at by an employee of ETG, but they may still have more errors.
3. Lab 5(UART), Lab 8(Servo), Lab 2 (Movement), Lab 4(Buttons)
4. LCD
    a. The LCD is very complex and optimization is not a high priority compared to the other labs

3.5.2 Integration Testing/Automated Testing

1. DSS (Debug Server Scripting)
    a. Be able to run a script that can load code onto the microcontroller and run a simple program. We used the bash script to build the code and use DSS API to make breakpoints to print out the result.
2. Continuous Integration

    a. Use Gitlab (inside of CI/CD tool) to trigger a DSS script when changes are added to the git bash commit and push to the git repository. We added the register git runner at window command.
3. Entire test pipeline
    a. On changes to the git repository, testing for each lab is automatically run and results of the tests are logged and reported. We used the python code to show the test result that is failed or passed.

3.5.3 Simulation of the external hardware

1. Create virtual Ping sensor
    a. The ping sensor is the simplest piece of hardware to simulate. It will be a good option to pick to prototype our simulation solution.
2. Create virtual IR sensor, Servo, UART
    a. These three devices are the next two simplest hardware devices to simulate. The IR sensor will require being able to send back an analog signal, most likely requiring a DAC. Because the servo does not send feedback to the microcontroller, we will need to implement a way to observe it.
3. Simulation of the Roomba, LCD
    a. These two devices are very complex, and it may not be feasible to simulate them in full. A partial simulation may be needed to test certain labs.

3.5.4 Git repository

1. Remove any build errors with current codebase
    a. Old configuration settings and file include issues need to be worked out
2. Remove unnecessary folders/files
3. Create 2 repositories, One for professors/TAs, and one for students
    a. Upload all necessary files for these two repos

3.6 Project Tracking Procedures

We will utilize Gitlab built in issues tracker to keep track of all tasks that exist, and which team member is assigned to which task.

We will also have bi-weekly status reports where we report the work we have done, and the hours spent over the previous two weeks. This is how we will make sure each team member is working hard and contributing to the project. Since COVID-19, we have used virtual meetings with not only advisors or professors but also team members.

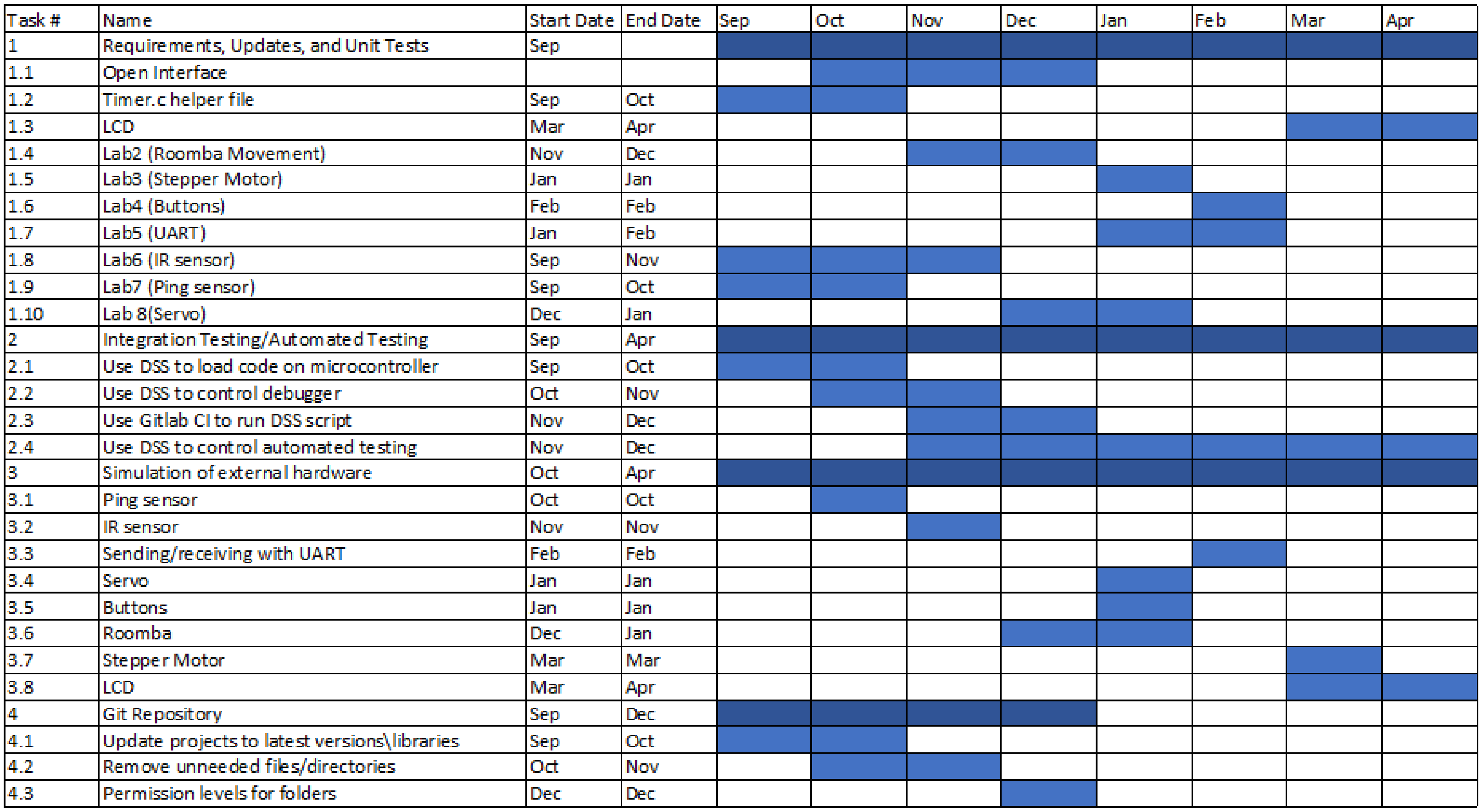
3.7 Expected Results and Validation

The desired outcome is tests that cover all the code inside the libraries for each device.

Because the code base for each device is small, and each of the devices are mostly independent from one another, having tests for the high-level requirements for each lab will confirm that our solutions are correct.

4. Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline



Task #	Name	Start Date	End Date	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr
1	Requirements, Updates, and Unit Tests	Sep		🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
1.1	Open Interface				🟦	🟦	🟦				
1.2	Timer.c helper file	Sep	Oct	🟦	🟦						
1.3	LCD	Mar	Apr							🟦	🟦
1.4	Lab2 (Roomba Movement)	Nov	Dec			🟦	🟦				
1.5	Lab3 (Stepper Motor)	Jan	Jan					🟦			
1.6	Lab4 (Buttons)	Feb	Feb						🟦		
1.7	Lab5 (UART)	Jan	Feb					🟦	🟦		
1.8	Lab6 (IR sensor)	Sep	Nov	🟦	🟦	🟦					
1.9	Lab7 (Ping sensor)	Sep	Oct	🟦	🟦						
1.10	Lab 8(Servo)	Dec	Jan				🟦	🟦			
2	Integration Testing/Automated Testing	Sep	Apr	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
2.1	Use DSS to load code on microcontroller	Sep	Oct	🟦	🟦						
2.2	Use DSS to control debugger	Oct	Nov		🟦	🟦					
2.3	Use Gitlab CI to run DSS script	Nov	Dec			🟦	🟦				
2.4	Use DSS to control automated testing	Nov	Dec			🟦	🟦	🟦	🟦	🟦	🟦
3	Simulation of external hardware	Oct	Apr	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
3.1	Ping sensor	Oct	Oct		🟦						
3.2	IR sensor	Nov	Nov			🟦					
3.3	Sending/receiving with UART	Feb	Feb						🟦		
3.4	Servo	Jan	Jan					🟦			
3.5	Buttons	Jan	Jan					🟦			
3.6	Roomba	Dec	Jan				🟦	🟦			
3.7	Stepper Motor	Mar	Mar							🟦	
3.8	LCD	Mar	Apr							🟦	🟦
4	Git Repository	Sep	Dec	🟦	🟦	🟦	🟦				
4.1	Update projects to latest versions\libraries	Sep	Oct	🟦	🟦						
4.2	Remove unneeded files/directories	Oct	Nov		🟦	🟦					
4.3	Permission levels for folders	Dec	Dec				🟦				

Figure 4.1: Project

This is our proposed timeline for the project. Some tasks may take longer than planned/become delayed. This largely depends on how many unexpected issues are uncovered.

4.2 Feasibility Assessment

Our team should be able to deliver updated versions of the device libraries that satisfy all requirements. A challenge of the project is that it is nearly impossible to find every possible bug in the code. We cannot guarantee every issue will be uncovered and solved.

4.3 Personnel Effort Requirements

Each team member will contribute a minimum of 6 hours per week to the project.

Task#	Task	Description	Hours
1	Requirements, Updates, and Unit Tests		Total: 138
1.1	Open Interface	Verify the openinterface.c/.h files are written correctly based upon its datasheet, all open interface commands are successfully sent to the Roomba, and sensor data sent back from the Roomba is parsed correctly.	20
1.2	timer.c helper file	Verify the timer.c/helper files are written correctly based upon its datasheet, and that functionally work at Code composer.	15
1.3	LCD	Verify the lcd.c/.h files are written correctly based upon its datasheet, and that all characters can be displayed correctly on all 4 lines of the LCD.	20
1.4	Roomba Movement	Verify the code for the Roomba movement identifies forward/back, left/right correctly for each of the commands.	15
1.5	Buttons	Verify the code for the buttons identifies pressed/unpressed buttons correctly for each of the four push buttons.	8
1.6	UART	Verify the UART correctly communicates with Putty, that all sent values can properly appear on the LCD, and echoes back to putty work properly	15

1.7	IR sensor	Verify the code are written correctly based upon the datasheet information and the code should function properly.	15
1.8	Ping sensor	Verify Correct distance is reported, and the registers used in the lab are set correctly.	15
1.9	Servo	Verify servo moves to the correct position given a desired angle, and the registers used in the lab are set correctly.	15
2	Integration Testing/Automated Testing	Run the tests automatically on a commit to the git repository. Use GitLab's built in CI tools	Total: 85
2.1	Use DSS to be able to load code onto the microcontroller	To do this we can follow the TI's guideline to work with this set up. It is not difficult because TI gave us many guidelines for it. To set up the work environment, simply use script select command section at Code composer studio.	30
2.2	Use DSS to be able to control the debugger	Use the TI Scripting API to implement the breakpoint to control the debugger to print out the result at the command prompt. Wrote the JavaScript to configuration the script code at DSS code composer studio. This will help later to build the code at GitLab's continuous integration work.	5
2.3	Use GitLab's built in CI platform to trigger a DSS script when changes are made to the git repository	It is already given at GitLab. We need to set up the git runner to make it continuously test our code whenever we push the commit to Git. Add the yml file to set up the jobs that we want to load and compile it and make some command to run it during the compile work.	20
2.4	Use DSS to assist in creating automated unit tests for each point in 3.3.1	We figured this out to use the Batch script language. So, we created a bat file which is included with the command (Because we will use	30

		this command several times) It helps us to use Window command to run this work at the same time.	
3	Simulation of external hardware	Simulation code running on a separate microcontroller that mimics the actual sensors/actuators.	Total: 55
3.1	Ping sensor	Detect a send pulse and send the distance pulse back which corresponds to the currently set distance. Allow the currently set distance to be changed by the connected microcontroller.	3
3.2	IR sensor	Detect an object and send the voltage signal to ADC to calculate how far an object is.	10
3.3	Sending/receiving with UART	Send characters to the Cybot and echo back to Putty, ensuring data sent is correct	2
3.4	Servo	Detect a PWM wave and set the servos position value to the corresponding angle. Also allow the current servo position to be read by the connected microcontroller.	8
3.5	Buttons	Be able to press/unpress buttons. Allow buttons to be pressed/unpressed by the connected microcontroller.	2
3.6	Roomba	Be able to receive open interface commands and send back a packet containing simulated sensor data.	30
4	Git Repository	Set up two Git repositories, one for group and TA's and the other one for students, it will be available to manage different files in each repository. We decided to separate the two modules. One for	Total: 15

		professors and TAs(Private), others for the students(public).	
4.1	Update projects to the latest version of Code Composer/Latest libraries. Also filming the each of lab demo videos	Ensure the code is correctly updated to current versions. Also, show the demo steps for latest labs.	20
4.2	Remove unneeded files/directories	Review the directory and files that we need or not. If it is not required from our project, we can remove it.	2
4.3	Be able to change permissions levels for different folders.	Allow students to access code provided for them without letting them access solution code	3

4.4 Other Resource Requirements

- 2 TI TM4C123GH6PM microcontrollers for the testing system
- 1 MCP4725 DAC for creating analog signals for the IR Sensor simulation

5. Testing and Implementation

The nature of our project is such that most of the work is to support testing. Because most of the device libraries are independent of each other, our tests are mainly unit tests.

5.1 Interface Specifications

We are creating a simulation of all of the external hardware (IR sensor, sonar sensor, etc.). This allows our code to interface with "the outside world" in an automated way. We won't need to manually retest everything after a code change.

5.2 Hardware and software

5.2.1 Code Composer Studio (CSS)

Code composer is the IDE TI provides for development on our microcontroller. It includes an advanced debugger and many other utilities that assist with development and debugging. We used this for code development, manual testing, and research into problems in general.

5.2.2 Debug Server Scripting (DSS)

Debug Server is based on debug engine of Code Composer Studio. It can access to not only DSS and the Code Composer IDE. It provides a JavaScript console and a Java API to load code onto the robot, set breakpoints, change variables, and other debugging features. We used this primarily for automated testing.

5.2.3 Simulation of external hardware

In order to simulate the external hardware, we will use a separate microcontroller to run simulations of each device. The main microcontroller will be able to query the simulation program to request its current state (i.e. to inspect the servo position) or to set distance values for the distance sensors.

5.2.4 GitLab built in Continuous Integration tool

Gitlab includes built-in CI (Continuous Integration) tools that allow us to automatically run our tests on a commit. It also displays the results of the different stages of the CI pipeline (Build and Test).

5.3 Functional Testing

Our testing is two-fold, manual and automated. Unit tests for each of the device libraries will have automated unit tests.

Other parts of our project will only be manually tested, either due to impracticality of automated tests or lack of time.

5.3.1 Unit Testing

Each .c file will have tests that verify the requirements given in either its associated lab manual or required by those labs (i.e. Open Interface, Timer). This includes tests for proper register initialization and tests for system functionality.

Device/File	Items to test
Timer.c	● waitMillis and waitMicros wait the correct amount of time. ● getMillis and getMicros collect the correct amount of time.
Roomba	● Correct control messages are being sent to the roomba ● packets sent by roomba are being parsed correctly
LCD	● test can be written to all four lines of the LCD ● each possible character is displayed properly

Sonar(PING) sensor	● Correct distance is reported back accurate from 2-300cm ● Register initializations are correct
IR sensor	● Register initializations are correct ● Correct quantized value is sent back based on input voltage
Servo	● Servo can move to correct angles 0-180 degrees ● Register initializations are correct
Push Buttons	● correct readings of button pushed/not pushed ● Register initializations are correct
UART	● All possible characters can be sent and received ● Register initializations are correct

5.3.2 Integration Testing

We have Continuous Integration using Gitlabs built in CI tools. When we add the configuration file (.gitlab-ci.yml), this file is very important and has detailed instructions on what to do to build, test, and deploy. These instructions are carried out by the runner(s). Instructions are organized by stages. So, a pipeline is composed of stages. A stage consists of jobs. These jobs are carried out by specific runners. Since we have many lab codes that we make some changes or test it out, we cannot manually retest everything for each single change we make. Also, if someone makes changes and does not have time to fully test, we will have broken code at our repository and users will not be able to access code that runs correctly. So, CI will help us to not only reduce our testing time but also reduce the errors from bugs.

5.3.3 System Level testing

Because each of the device libraries are small with only a few functions, some of our unit tests will also function as System level tests. Therefore, we will consider Unit tests and System level tests as part of the same group for our project.

5.3.4 Manual Testing

Not everything is able to be automatically tested. Due to lack of time, the simulation code does not have any test written, it is only tested manually.

5.4 Process

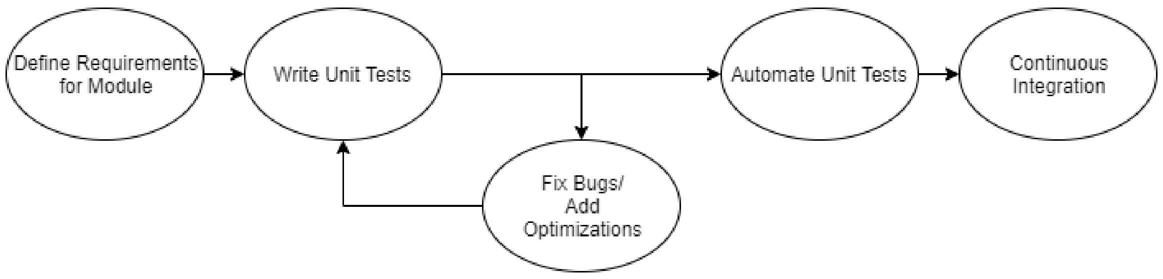


Figure 5.1: Testing

Our process starts with defining the requirements for each device and then writing unit tests to verify those requirements have been satisfied. Although the diagram above is mostly linear. In reality there will be constant iteration, with new requirements possibly being added as the year goes along. we may go through this process multiple times for each module.

For manual tests, the process will be slightly smaller. We will only verify the functionality once, and then document those results and processes if anyone intends to test them again in the future.

6. Closing Material

6.1 Conclusion

All in all, we have not only developed testing but also completed our project goals. We finished automated testing on our microcontroller, continuous integration with Gitlab, Simulation of our eternal sensors and other hardware, Git repository organization, and have documented our final progress.

6.2 References

Manuals used:

TI TM4C123GH6PM microcontroller datasheet

Sharp Infrared distance measuring sensor GP2D12J0000F datasheet

Parallax PING))) Ultrasonic distance sensor 9#28015) Datasheet

Parallax standard Servo (#900-00005) Datasheet

iRobot Create v2.0 Open Interface specification

GitLab Runner Doc

Texas Instrument DSS Doc