4 Testing

Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, power system, or software.

The testing plan should connect the requirements and the design to the adopting test strategy and instruments. In this overarching introduction, given an overview of the testing strategy. Emphasize any unique challenges to testing for your system/design.

4.1 Unit Testing

What units are being tested? How? Tools?

The units being tested are the widgets within our application and the helper functions included in our server. Any unit of code utilized where the input and output are both from and within the same area of the code base can be unit tested. In Flutter unit testing has provided tooling. Docker provides an automated way for our team to execute our unit tests. We will use unit tests on the functions that we create to alter or interpret our data within an enclosed environment.

- The motors will be tested by hard coding in various movements for the bots to test its response, speed, agility, etc.
- The bump sensors will also be tested with print() methods to the console to ensure that bumps are detected.

4.2 Interface Testing

What are the interfaces in your design? Discuss how the composition of two or more units (interfaces) are being tested. Tools?

Interfaces for our robots will include a Raspberry pi over internet connection to a controller on a desktop. The pi will be sending data such as controls from the application to the bots motors. As well as, the bots sensors sending data to the application and motors for control.

Testing of the application will be done by sending coded data for controls, buttons, over a wireless connection and seeing what errors appear when any other buttons are pressed. As well as using tools like flutters automated testing features for widget testing to check for interface correctness on all UI items displayed to users. Other testing would include displaying data from database to user or using stored database data when sensing.

4.3 Integration Testing

What are the critical integration paths in your design? Justification for criticality may come from your requirements. How will they be tested? Tools?

//software using flutter integration testing

Each command in our in user application will be tested to ensure that the correct data is transmitted between app and bot.

- When the user inputs a command to drive the bot, we will check to ensure that the bot is receiving the correct signal using print() methods to the console
- We will ensure that the camera is able to stream to the server, and the server is able to be accessed by the user via the application
- We will ensure that the arena camera is able to use machine learning to detect objects and relay their positions and orientations to the server

4.4 System Testing

Describe system level testing strategy. What set of unit tests, interface tests, and integration tests suffice for system level testing? This should be closely tied to the requirements. Tools?

Overall test

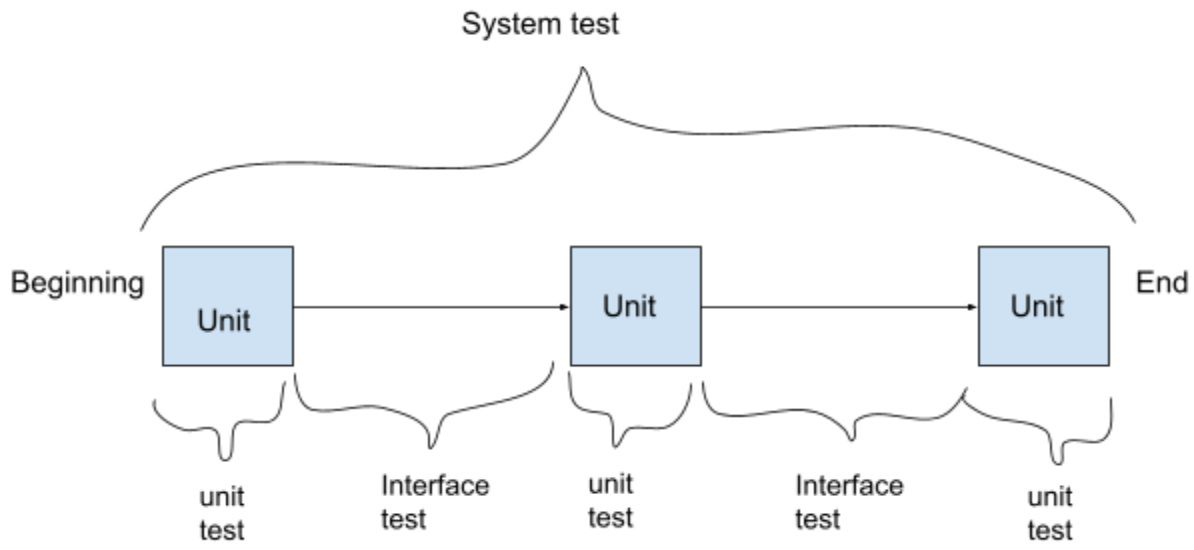
This set of tests spreads through each unit and therefore tests the overall abilities of unit tests, interface tests, and integration tests. A basic input will start the test and each unit will deliver data to the next unit until the final output is obtained. If the system test fails, then further individual testing will be done to check each section for the failure.

One example of a system test is to push the ‘drive forward’ for two seconds. The unit software is tested for detecting the command, then the interface to the robot is tested for ‘to bot communication’ then the unit software on the bot is tested for detecting the signal and supplying an output to the wheels, which tests the unit wheels and hardware of the bot. The bot movement tests the Machine Learning object detection unit which in turn tests the interface back to the user. The final output will be movement detected by machine learning.

Benefits to this test include being able to check practical time delay by starting a timer when the input is pressed and stopping, when the output is received.

Another benefit is this checks the shared variables of the individual tests, such as physical space, memory, and CPU runtime.

If the testing passes, then more extensive practical testing can begin.



4.5 Regression Testing

How are you ensuring that any new additions do not break the old functionality? What implemented critical features do you need to ensure they do not break? Is it driven by requirements? Tools?

Our main tool for handling regression testing will be Git. Git allows each individual programmer to start a new branch whenever they begin a task, whether it be adding a new feature or a bug fix. In this way the master branch is completely unaffected by the new code while it's still being worked on, meaning everything will continue to be available in its previous state the whole time the new branch is being worked on, even if the person working on that branch accidentally breaks any of the features with their changes. Git also allows you to set up automated tests that run when merging your branch back into master which can catch if certain things were broken even if the programmer themself hadn't realized it. In the event that a game-breaking bug does manage sneak by undetected until it is a part of the master branch, Git allows you to revert to any of your previous commits, so even though you may lose a lot of your progress, you will always be able to go back to a working version once it exists. Finally, if two people make different changes to the same code, Git won't allow you to merge until the conflicts are resolved, meaning it will catch scenarios where your code works perfectly when tested on your branch and another person's code works perfectly when tested on their branch, but they would fail in conjunction with each other because of a dependency on something that no longer exists in the format you expected once the other person's work is considered. With the power of Git, if a new feature or patch would break an existing feature, you can always choose which version you would prefer to have. This means we can prioritize versions where the core gameplay mechanics are intact at the expense of extra bells and whistles like a post-goal, celebratory dance should such a situation arise, regardless of which version is newer.

4.6 Acceptance Testing

How will you demonstrate that the design requirements, both functional and non-functional are being met? How would you involve your client in the acceptance testing?

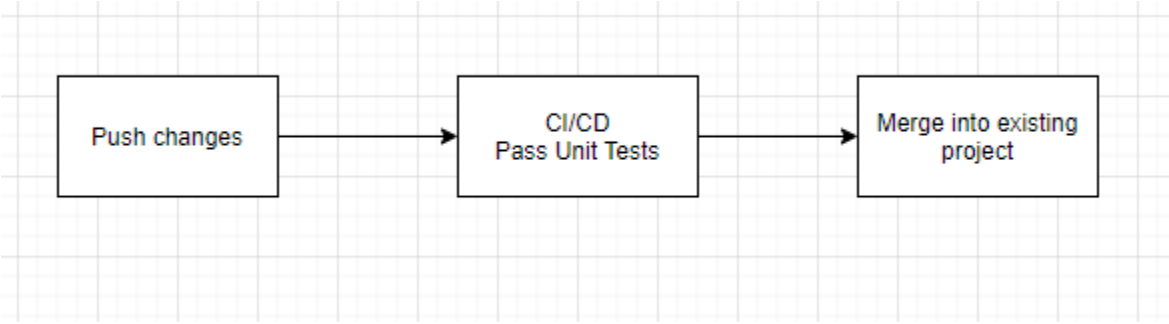
Being that our end product is designed to be a game, it may be best for many of our functional and nonfunctional requirements to be measured by third party input. To test our usability and UI/UX requirements, we should see how well users learn the game controls without assistance. This can be measured either in a guided way by timing and observing, and/or by asking the user about their experience with the controls and interfaces via survey.
With the nature of our project being a game, our functional requirements should primarily be covered by these user tests/demos. This data can be collected via survey. Additionally, nonfunctional requirements such as the aesthetic appeal of the bots and application can be inquired about in the survey to see how our product stands up.
Our client can 100% be involved in the acceptance testing just by playing the game, driving the bots, interacting with our application, and providing feedback. Our acceptance testing should take place over several demos with our client and/or friends outside of the project so that we can improve our end product with each iteration of feedback.

~~4.7 Security Testing (if applicable)~~

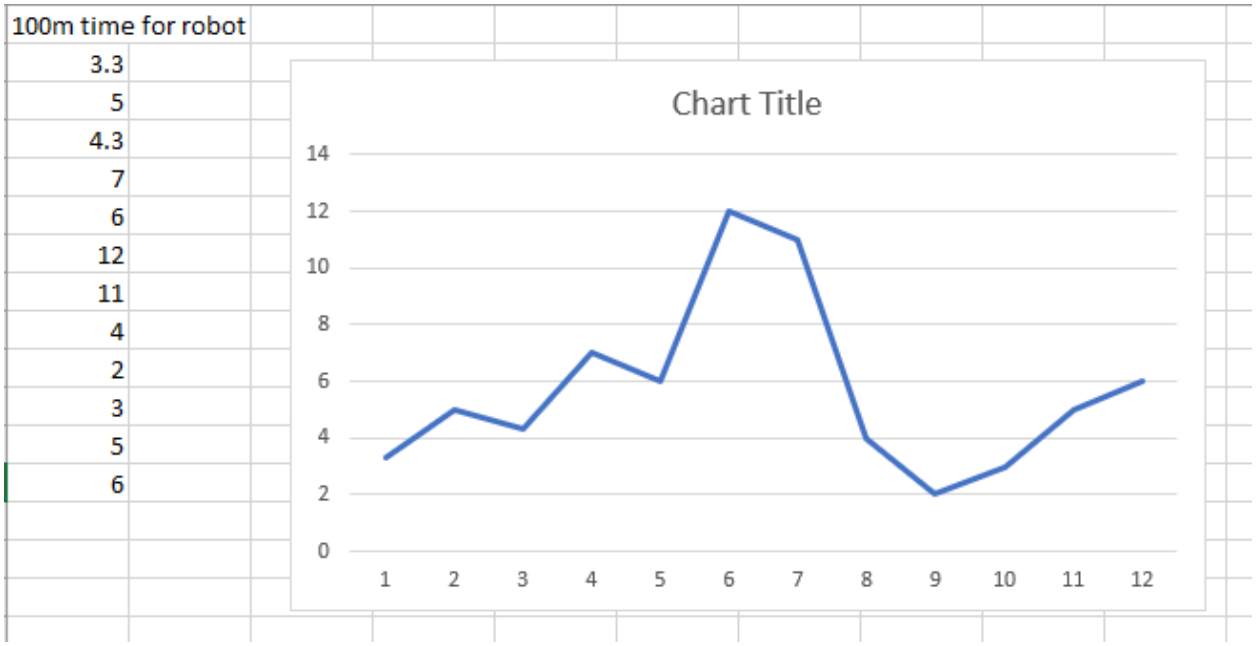
4.8 Results

What are the results of your testing? How do they ensure compliance with the requirements? Include figures and tables to explain your testing process better. A summary narrative concluding that your design is as intended is useful.

We are expecting our results to reach our expectations. We can ensure compliance with the requirements by testing our qualitative requirements to meet those expected projections.



This will be how our code testing process will be and for UI stuff we will test it through human use.



Hardware will meet our requirements by measuring and making sure the average is within our requirements. We could either do this via excel or make the robot meet those expectations on multiple runs.