Robot Dummy Testing & Procedures

Jonathan Lu, Brandon Huynh, Jaiden Pickford, Raphael Talento

June 10, 2024

Abstract

The purpose of this report is to document and outline all testing procedures of the robotic dummy platform. These will outline the exact safety steps for what anyone working on the robot should either follow or keep in mind of. We'll be first explaining what exact testing as been used on the robot. After this, we will use another section for procedures on using the robot during any future testing.

1 Introduction

Testing and outdoor development of the robot was conducted by all members of the undergraduate team during the following testing periods: 05/27/2024 to 05/31/2024 and 06/03/2024 to 06/05/2024. While most code and connectivity was throroughly tested before, we needed to understand what exact real-life behavior would be exhibited by the robot. Additionally, we needed to gain some insight on what proper steps one should take when using, testing, or developing the robot. There will be two sections to this report: testing and procedures. Testing will outline what significant testing and behavior we have tested for and allowed the robot to show. Our hope is to show what the robot can do as well as open up future improvements. For procedures, we have included this section for safety and for general use of the robot that we have gained from testing it.

2 Testing

Here, we will be showing what significant tests that we have performed on the robot. Each test will be brief to explain the behavior, how to replicate the test, and open up a possible fix for the future.

2.1 Basic User Movement

From the app to the robot, we tested basic user movement to move the robot forward, left, right, and backwards. As far as movements go, the robot will properly respond to which direction that the user would like to move the robot to. The GUI of the app for movement can be found here: To replicate movement, simply press any of the directional arrows to the right of the screen. Make note of the direction of the arrows that correspond to the direction that the robot will move to:

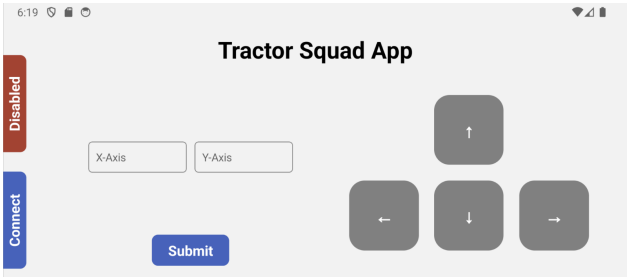

Figure 1: App UI

- Up arrow = move forward
- Down arrow = move backward
- Right arrow = move forward to a slight right curve
- Left arrow = move forward to a slight left curve

While the basic movement worked between the app communication and motor movement, there was one quirk that we noticed. Due to the nature of BLE and how it is handled on both the app and Pi side, there is a bit of a delay. We timed this delay to about 2 to 2.5 seconds. How the delay looked was that if a user quickly tapped, for example, the up arrow on the app, the robot may move immediately due to the forward signal, but the following stop signal will take around 2-2.5 seconds to finally let the robot stop. This means that a quick tap will appear as the robot moving for a couple seconds before stopping.

However, we noticed that the aforementioned delay is consistent and always around 2 to 2.5 seconds. Therefore, if a user holds down the up arrow to make the robot move forward longer than 2 to 2.5 seconds, then letting go will immediately stop the robot. We are assuming that BLE with either the app or Pi is making a queue of requests send from the app to the Pi. We are thinking that this queue takes around 2 to 2.5 seconds to finally finish, thus causing a delay that can appear with quick movements but not be noticeable with holding down for long movements. While we did not have time to implement this, we believe that creating a separate queuing system could fix this delay by incorporating it into the app. The queue may be network/bluetooth sided, and so if the app locally has its own queue, it isn't clogging up the network which can alleviate some delay.

One thing to mention is that all app buttons operate on an onPress and onRelease basis, meaning that behavior is signaled out based on pressing the button and releasing it. Inherently, React Expo/Native makes these buttons impossible to press simultaneously. This means that a user is not able confuse or break the robot by holding down all buttons at the same time, which is good from a user and safety perspective.

2.2 Bluetooth Connectivity and Enable/Disable Buttons

Of course, due to the robot responding to user movement, we can confirm that Bluetooth works for the phone to connect to the Pi. As mentioned before, there is an inherent delay. One thing to note, the app is programmed to only find Bluetooth devices that are named "tractorsquad". The Raspberry Pi that we are using with the robot, or specifically the SD card in it, already has that name set.

To make the Pi a peripheral that a phone with the app can see, it should have a bash script that runs our Bluetooth program/driver and movement program/driver upon Pi startup. It may take a couple seconds for the phone to find the Pi, and it may take a few app restarting for it to show. Bluetooth connectivity will be successful when the app UI changes the "connect" button to "disconnect" and when the "Tractor squad app" changes to "Please select GPS or user movement". Due to safety concerns, we have our movement program detect for the enable signal from the app before allowing movement to the motors. How a user can do this is by pressing the red button on the left side labeled "disconnected". A prompt will show up to confirm turning on the robot. Selecting no will do nothing except close the prompt. Selecting yes will enable the robot which can be seen by the red button on the app turning green and displaying "Enabled".

Once the enable signal is sent, the robot should respond to both gps and user movement, and a quick press of the green button should fully disable all motor movement signals from the app, which shows no prompt.

We found that all behavior from this is working as expected with probably a slight delay with the enable button, where a user may need to wait 2 to 2.5 seconds after pressing enable for a movement button to operate. One thing that we wanted to add in was a simultaneous button press for the enable button with a movement button. This would be a safety feature to prevent people from hitting a movement button on the robot on accident. This can be done in React Native, although it would require redoing our current configuration which we had no time remaining to do. This would be an overhaul if further development continues with the robot.

2.3 GPS Movement

GPS movement is still a little inaccurate with movement. However, the app should be able to send x and y coordinates to the Pi by typing them into the two left-side text boxes and then hitting the "Submit" button. One thing to note, while the user can type in any character, the Pi will only interpret numbers, and it will break upon entering anything else besides a negative or decimal point. The Pi will only interpret 6 significant figures, and it will ignore anything past 6 significant figures. This is due to Bluetooth Low Energy limitations on how much data we can send at a time, so 6 sig figs is the limit.

The movement program on the Pi and its GPS module do not work as accurate as we'd like. Before ensuring that the robot will do GPS movement, ensure that the GPS module in the Pi Chassis is connected. It will connect when it stops blinking. Once that is confirmed, the GPS will try to determine its and the robot's position, and then try to calculate how it will reach from point A to point B. A startup sequence will happen during this process, where the robot will make a path within its primary path to determine its position and heading. It then uses that to interpret which direction it will go and how far it will go.

Due to the inaccuracy of the GPS, the Robot will sometimes fail to go the full distance to its destination. However, from multiple testing, we were able to confirm that the Robot will move into the correct general direction for its destination point. This is usually within a 10 to 15 degree margin of error.

To fix this, we ran out of time but had ideas to purchase a better GPS module because it has an inherent margin of error of 3 meters, which is huge. Another thing we could do with the GPS is use a bigger startup sequence to hopefully retrieve more GPS data to make a more accurate modeling for its starting direction/position and move to the proper destination. We believe implementing both aspects would improve the GPS accuracy, at least with letting the robot move to the actual correct position instead of falling short. This would also lessen the heading margin of error (10 to 15 degrees).

2.4 Heartbeat Bluetooth Acknowledgement

Our Bluetooth programs on the phone app and Pi has a sending and receiving heartbeat functionality. This heartbeat would be a series of acknowledgement signals sent from the app to Pi in order to confirm that the connection is still being made between both devices. The Pi will keep track of these signals, and if it loses the signal around 5 to 10 times, then it will shut off power to the motors to prevent them from moving due to a severed connection.

For testing purposes and due to the Bluetooth connection getting clogged up, we had to disable this feature, but it still exists in the code for both the app and Pi. They are currently commented out, but they can be uncommented to make the functionality work again. This may introduce a longer delay/lag between phone and Pi communication.

2.5 Emergency Stops

Hitting the emergency stops on the robot will severe only the power going to the motors. Power going to the Pi and motor driver/controller is still there. However, this will stop all movement as intended, which makes it great for testing and doing any E-stops. There is one quirk that exists due to how power is cut off with the emergency stops, and it is that any commands in the motor controller will still exist even if the motors are not moving. This is because the Pi and motor controller still have their power, so the Pi can still tell the motor controller what to do despite the motors being effectively off. This quirk and its procedures will be explained more in the "Procedures" section. But for now, using E-stops will properly shut off the motors.

2.6 Button spamming

Button spamming can occur with the app. As explained before, the Bluetooth connection between the app and Pi introduces a queue of signals to send over. Meaning, it could send a queue of "Forward", "Stop", "Right", "Stop", "Backwards", "Stop", and the robot will adhere to that movement, with a slight delay in between. This means that the robot could go forward for 2.5 seconds, stop for 2.5 seconds, go forward then right for 2.5 seconds, stop for 2.5 seconds, and so on. Now, with button spamming, this could introduce a whole series of signals being sent over. This is not necessarily an issue except for the delay with the Bluetooth. Due to the delay, one could be waiting a while for the robot to stop what it is doing.

A fix for this would be fixing the inherent delay with the Bluetooth. Please refer to the "Procedures" section if a spam of buttons was commenced and the user wishes to stop the robot from moving.

2.7 Disconnecting Bluetooth During Movement

There are a couple disconnection scenarios that we tested out. We tested trying to press the disconnect button on the app while holding down a movement button, and we tried running the robot out of range.

For the first test, we confirmed that a user is unable to disconnect the robot while holding down a movement button. This is due to how we designed the buttons on the app, where none of them can be pressed simultaneously. Due to this, a user would have to let go of the movement button to press disconnect, effectively sending the "Stop" signal upon releasing the movement button. This is great news to ensure that the robot won't continue running its last command since the Bluetooth connection is severed.

The second test that we tried also seemed to have passed. We noticed that the robot moved around 30 meters or 35 meters out before it stopped responding to user input. However, we noticed that every time it left that range, if the user was holding down a movement button then letting go after that range would properly stop the robot. This meant that the robot would be able to stop moving even if Bluetooth was severed. We are unsure of what is causing this since the signal is not sent until after letting go of the button, but we are guessing it is due to the queue and that it would be expecting a stop even if the connection is severed. Either way, it is good that the robot properly stops once it goes out of connection range and after a user has let go of any movement.

3 Procedures

This section serves as what proper procedures to make regarding using the robot. We have created this as a result of the testing that we have conducted with the robot. Most of the procedures involve stopping and starting the robot, so it will not be very long.

3.1 Turning on and off the Robot

Turning on the robot is a semi-lengthy process mainly for safety reasons. To turn on the robot, users will have to follow this sequence to ensure everything is on properly.

1. Turn on the robot (after unlocking Tag out/lock out) with any or all of the emergency stops **engaged**. When turning on the robot, users will need to ensure that the emergency stops are engaged so that any movements are not accidentally pressed or any previously saved movements do not start upon powering on. Once the power switch is engaged and the red light is on, wait around 30 seconds to as long as a minute to ensure the Raspberry Pi is on. Users can check on Raspberry Pi bootup by SSHing into it or checking on it using an HDMI cable and a computer. Once enough time has passed, users can move onto the next step.

2. Use a phone with the robot controlling app and open the app. Once the app is opened, click on the bottom left button labeled "Connect" and allow all Bluetooth and location permissions if necessary. Wait for the name "tractosquad" to show up on the list of Bluetooth devices and then connect. A successful connection will replace the text on the connect button to say "Disconnect". Do not do anything else just yet.

3. Ensure the robot is in a safe or ideal spot for movement before engaging any movement. Move the robot there with no one disengaging the emergency stops on the robot or enabling anything on the phone.

4. Once the robot is in a safe place, disengage any and all emergency stop buttons. Then, ensure someone (such as a spotter) is holding the robot in case of accidental or cached movement before pressing the start button. Have someone press the start button so that the green light turns on. If the robot happens to move, ensure that the spotter is able to hold the robot back or is able to reach an emergency stop.

5. If the robot is all good and the motors are ready, use the app and enable movement. To enable, click on the red button on the top left labeled "Disabled". When clicking this, a prompt will pop up confirming

if the user is ready to turn on the robot. If ready, confirm "Yes" and the button will turn green. Allow 2 to 2.5 seconds for the robot to acknowledge this.

6. Once everything is on and ready, ensure that there is a safe distance between users/observers and the robot, with only a trusted tester around to hit an emergency stop if necessary. Users can now use any movement buttons located on the right side of the app, or input any GPS movement to the left of the user-movement buttons.

7. Once testing is done, the app user must wait for the robot to stop by letting go of all buttons and waiting around 2.5 seconds. After, they must disable robot movement by hitting the green button on the top left labeled "Enabled" in order to disable the robot. After that, disconnect Bluetooth by hitting the bottom left button now labeled "Disconnect". Once all app communication is severed, someone should hit an emergency stop on the robot. Doing it in this order is good to ensure that the motor driver does not cache any previous movement in it, so that it won't continue previous movement upon next startup. However, if needed, users can hit an emergency stop before disabling app communication. Once this is all done, switch off the robot and initiate a tag-out/lock-out.

8. Ensure all emergency stops are engaged so that when the robot is used in the future, it is guaranteed to have all motors off. This is for redundancy and safety purposes.

9. Store the robot as appropriate. Do not charge the robot overnight if it needs power. It should be powered under someone's supervision to prevent any likelihood of a fire. This is just for a precaution, but it is important for the safety of everyone.

3.2 Emergency Stops

This procedure is to ensure that emergency stops are used correctly. When anyone hits an emergency stop on the robot, the only thing that gets cut off is the power to the motors directly. However, the Raspberry Pi and motor controller in the robot are still powered on by the batteries. Emergency stops should be used for any reason where the robot should stop moving such as but not limited to:

- The robot hitting a wall/object
- The robot hitting a user/observer
- Freak accidents
- Testing Purposes

There are four emergency stops on either side of the robot. At least one needs to be engaged to fully cut power to the motors. For safety precautions, the robot should always have all emergency stops engaged when it is not in use, so that they are engaged once the robot gets powered on. One simply needs to press on a button to engage it. Twisting the button will disengage it.

When an emergency stop is used, there can be a chance that a signal sent to the motor controller could still be "alive" when disengaging the emergency stops and powering the motors back on. What this means is that whatever previous signal that made it to the controller, it could still be running. Therefore if the motors receive power again, then the motors will keep doing that previous movement. It is advisable to wait a certain amount of time to allow the motor controller to receive its stop signal before turning the motors back on.

However, if the motors will not stop spinning, then users should perform a **power cycle** on the robot to clear the signals from the motor controller. To do this, simply make sure the emergency stops are engaged on the robot and everyone is in a safe position. Once this is done, power off the entire robot using the red power switch. All LEDs should be off at this point. Wait 30 seconds to as long as a minute so that everything power cycles correctly. After waiting, keep all emergency stops engaged and power the robot back on so that the red light is on. Then, disengage all emergency stops. Before hitting the green button, have a spotter or someone at the ready to hit an emergency stop on the robot. Once a proper spotter is

positioned, have someone else press the green button to start the robot, in which the green light will power on. Due to the power cycle, the robot should not be moving since the motor controller's signals have been cleared. However, ensure that both the spotter and the person powering on the robot are safely expecting the robot to move if such an occurrence happens. Once you can confirm that the robot will not move on its own, users can continue testing or doing what they want with the robot.

It should be noted that when in doubt, hit the emergency stop on the robot and analyze the current situation before restoring power back to the motors.

4 Conclusion

This report serves as the documentation for how we performed testing and basic procedures to use the robot. We consider the information on this report to be like a living document, and even if testers are not able to write onto the original document, all text and information is free to be distributed into future documentation editions. This robot is considered a very early prototype, so all information and testing is expected to be added onto.