Emergency Stop Final Project

Jeremy Cook and Jessie Chen

May 2017

1 Abstract

Autonomous robots are not fully autonomous yet, and it should be expected that they could fail at any moment. Given the validity of this statement, there must be a fail safe measure to stop a robot in an easy and fast manner. To solve this issue, we created a wireless emergency stop button for the version 2 and version 3 segbots.

2 Introduction

We propose to introduce a new method to halt the robot with an emergency escape in the form of a big red button. When my partner and I were first learning about the segbots, we weren't fully aware as to how they functioned, and if they could cause damage to themselves or something else or even someone else if their code went awry. In many autonomous systems, there is an easy and understandable escape from the autonomous control back to manual control. This is the feature my partner and I wish to introduce to the segbots. Our prime objectives are to increase the safety of the robot and to simplify the user interface. As of now, if a user wishes to stop the robot, she/he has to position themselves behind the laptop and command a new 2D Vector pose to the Rviz window, or to kill the program responsible for actuating the motors of the segbot. Either of these two tasks can be very difficult while the robot is spinning and moving away from the user. My partner and I propose to create a wireless large red button, which is commonly known for halting a robotic operation thanks to television culture, which will be able to stop the robot at any instant.

3 Related Works

What happens when a robot's sensors stop functioning? It becomes uncertain of many quantities, and either gives up on its assigned task, or continues to function as if nothing wrong went at all, because it broke in the exact location that tells it when something goes wrong. In the paper by Stolt Andreas et al. [1], the project aims at identifying the loss of sensorimotor data and suggests a method of automatic detection that will prevent the robot from harming itself or other people/objects. The robot used in the paper is an arm that hits an emergency stop button when it extends past its local boundaries. The arm does not stop immediately however, but slowly ramps its speed down to give it a minimal jerk. This paper has also brought up the idea where in accordance with the remote button, we could have an automatic detection algorithm that will stop the robot when a dangerous situation is detected.

It's clear from Dhillon, Balbir S., and A. R. M. Fashandi. [2] that the most dangerous robots in today's world are industrial robots. In 1982, the world robot population was estimated to be at 30,000, and increased rapidly to 520,000 in 1992. However, as we progress to a more automated society, it's not hard to imagine more robots interacting with us in our daily lives, and in ever increasing risk. For example, it's safe to say that most humans trust an auto piloted airplane will carry us to our desired destination with a low risk of failure. We are however not at the point where we can remove the pilots from the cabin and fully trust an autopilot. Another interesting area of automation comes with driverless cars, or self driving cars. Many people are more wary of the failure rates of these systems, and will continue to be wary until death do them part. Besides the health hazard that artificial intelligence and automated systems bring into our lives, there is also an economic factor to be considered. When one test robot costs half a million dollars, it would be unwise to program it willy nilly without considering any potential damage to the individual parts of the robot. According to published literature in 1997 [2], the mean failure time was only 500 - 2500 hours. This means that if you had an industrial robot (in 1997) running 23 hours out of 24 hours in the day, it had a very high chance of failing within the first 4 months. At the end of the day, we have still not reached a point where we can rely on robots with human free interaction 100% of the time. However, I never think we will reach this point either. Robots will always start to fail at some point, and humans will need to intervene to fix them. "Generally speaking, a robot is blind, deaf, mute, dumb, and unconscious. The sum of these elements

render robots dangerous and unforgiving.“ [2]. The root cause lies in the lack of intelligence of the robot. Once (or if) we succeed in creating truly intelligent robots, we can program them to be aware of the humans in their surroundings and take caution in ways that we might not have imagined. I predict this date is long into the future, and until then we need a safety escape for every robot. This is our purpose of our project at its base, but can also be used as a blueprint for other button triggered events.

Work on creating a wireless emergency stop button may also lead to further research into implementing a more complex wireless remote. This remote may have more options in terms of controlling different functions of the segbots. For example, implementing a joystick may make tele-operations safer and easier to control. Since the remote is wireless, human operators can maintain a safe distance away from the robot, preventing human-robot crashes and injuries. A wireless remote may also be used to move the robot while the robot is not in reach in the case of dangerous situations and events. This also extends into a concept very similar to Amazon's dash buttons. The user could create a button whose only purpose is to fetch the user coffee, or maybe remind other coworkers of an upcoming meeting. Our project lays the groundwork for these implementations on the segbots, that have been developed elsewhere in the world already.

4 Mechanical Design

There are two sides to this project, the mechanical side and the software side- we will first dive into the mechanical aspect of this project. We ordered a simple Single Pole Single Throw (SPST) switch from Amazon in the shape of a typical emergency stop button. We then wired a 2S 7.4V Lipo, NodeMCU v1.0, and on/off switch to the button in order to make a button press be received wirelessly. Below is an image of the simple wiring schematic.

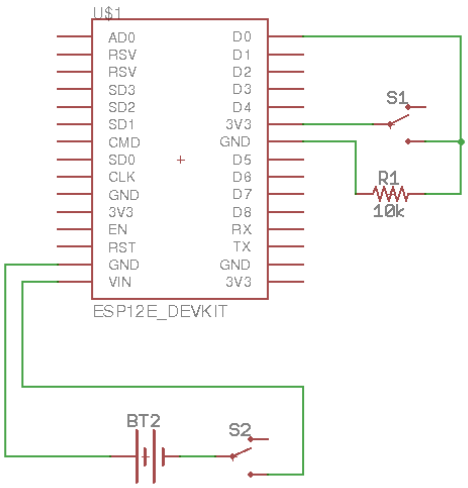

Figure 1: Wiring schematic of emergency stop button.

The button is wired to pin D0 with a pull down resistor attached to ground. Whenever the button is pressed the D0 pin is pulled high, and when the button is released the D0 pin is pulled low.

We initially intended for the NodeMCU, which is a WiFi based chip, to communicate directly with the segbots. However because the robots are connected to the network "utexas" which is a WPA2 Enterprise network, our design didn't work because the NodeMCU chip is not able to connect to WPA2 networks. After some thought, we decided to pivot our project to wireless communication between two NodeMCU chips. One would be inside the casing of the emergency stop button, and the other would be connected via USB with the segbot. The NodeMCU which is connected directly to the robot, which we will now refer to as the base station, would communicate with the robot via serial commands. In order for the two chips to communicate with each other, we programmed the base station to broadcast an access point with the name "emergency_stop" and a password. The NodeMCU in the emergency button, which we will now refer to as the client station, connects to this access point on a specific port and sends commands via UDP. A packet "stop" is sent out when the emergency button is pressed.

Average current consumption of the NodeMCU has been recorded to be 83 mA, so given our battery capacity of 800 mAh, we can get a crude estimate of how long our emergency stop button will last on a full charge with:

$$\frac{800\ mAh}{83\ mA} = 9.64\ hours \tag{1}$$

As was described above, the button cannot function without the server to handle the button signals. Below is an image of both the server and the button.

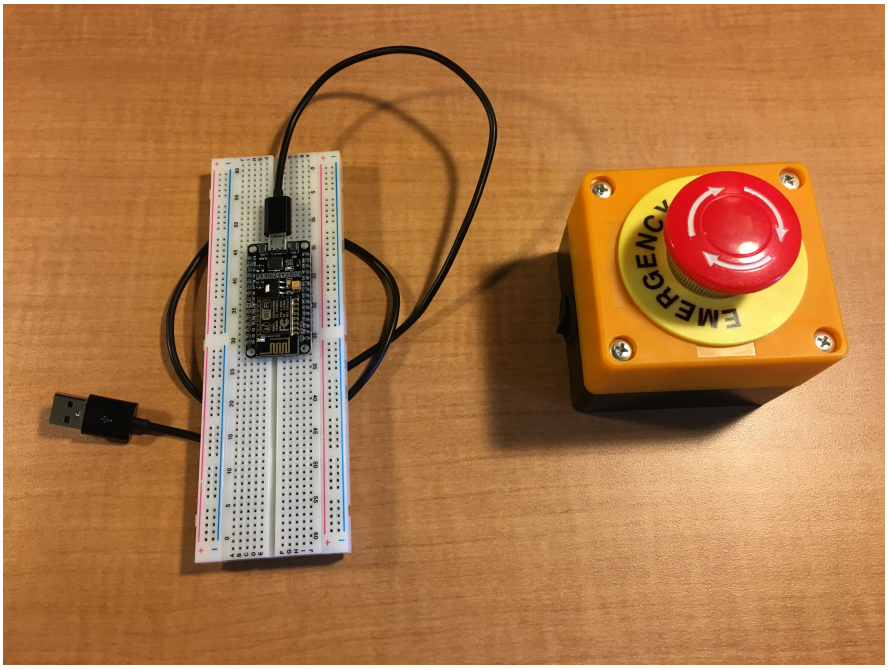

Figure 2: NodeMCU server on the left, and the stop button on the right.

5 Software Design

The robot has two main methods of operation: autonomous and manual. Here is the list of different modes of operation that we need to stop:

- Single navigational goal
- Teleoperation
- KR execution Task
- Custom node

- Nodes publishing to /cmd_vel

Let's begin with a single navigation goal. These type of actions arise when a 2D navigational goal is requested in rViz after having localized the robot. Cancelling the goal is fairly straight forward as shown in the sample image below.

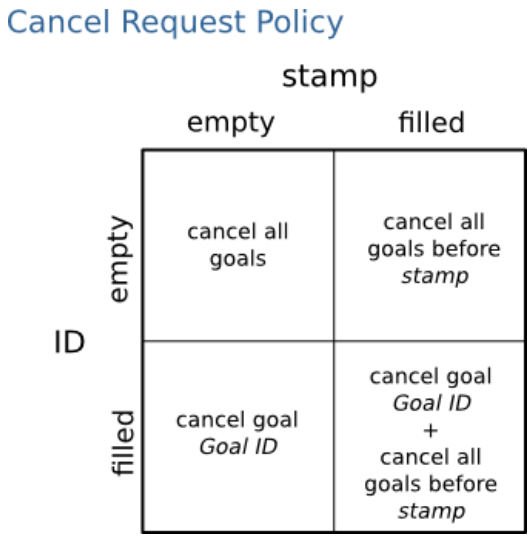

Figure 3: Cancel request policy courtesy of wiki.ros.org

To be safe, we want to cancel all goals, which corresponds to publishing an empty set in a string ("{}") to the topic "/move_base/cancel" of the form actionlib_msgs/GoalID.

Next we want to look into cancelling teleoperation commands, which requires a little more functionality. We simply can't publish a cancel goal message because teleoperation doesn't operate using goals but instead it operates by directly sending velocities to the wheels. In order to stop the mode of operation we have to kill the node responsible for publishing velocity commands and send a new velocity command of 0. To kill the proper node, we can use system command calls in C++ to directly execute commands in the terminal. First we run "rosnode list", to check which nodes are active. This returns a string of all running nodes, and if we find the string "/teleop_twist_keyboard", then we can kill it with another system call of the form "rosnode kill teleop_twist_keyboard". Once the node is killed however

the robot will continue to move in the direction of its last velocity command, so we publish a new velocity command of 0.

Next on the list of programs to stop are bwi_kr_execution tasks. These tasks include visiting a door list or message delivery or looking for a person. It is likely that on one of these tasks a student could open the door for the robot to the landing which contains the stairs in the GDC, in which case the robot would need to be stopped immediately. Similar to stopping the robot during the teleoperation mode, we search the list of running nodes for "/action_executor" and "/bwi_kr". These nodes are responsible for issuing the robot new goals and velocities, and must be killed without mercy. Once they are killed, we send the robot a new goal of its current position to halt any motion leftover from the task.

Second to last on our list of operational modes of the robot is a custom node. Without having to edit the code, a user can pass the name of their node they want to kill as a parameter into our main stop_base node. Once the stop button is pressed, the program looks to see if the custom node is running, and if it is, it kills it. As a safety measure the program then publishes a new velocity of 0 and sends a goal of the current position of the robot. We must do both because the program does not know a priori whether the custom node publishes velocity commands like the teleoperation mode, or if the custom node sends goals to the robot, like a 2D navigational goal, or if it does both. In any case, the robot will stop on the spot.

Finally, in order to kill any nodes which are publishing to the topic /cmd_vel, we run a system call in the terminal to see which nodes are publishing to the topic. Then we parse the string result to kill any nodes which are listed as publishers.

6 Evaluation

Initially there was concern as to whether setting the velocity of the robot to zero would cause the robot to tip over, but after extensive testing on both the version 2 and version 3 robots, we found that there was no cause for concern of the stability of the robots.

We evaluated the emergency stop button in all the test cases described, and found the button to work correctly. The scenarios where the button would not work is when the button is out of range of the robot, there is a connectivity issue, or the robot's control computer is frozen and is not

subscribing to new ROS messages. We did not choose to stop the robot when the button became out of range or was not connected because of the inconvenience it would bring to day to day operations. Rather, we wanted the stop button to be an optional addition to the robot, which could be used on demos or when testing new code.

7 Conclusion

After many weeks of work, we have created a successful emergency stop button for the version 2 and version 3 segbots at the University of Texas at Austin. Although this project was not cutting edge technology, it is important to not trade in safety for the shiny and expensive new features. A robot must be built on a solid and robust foundation to operate smoothly and to be easily built on in the future.

References

[1] Stolt, Andreas, et al. "Force controlled assembly of emergency stop button." Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011.

[2] Dhillon, Balbir S., and A. R. M. Fashandi. "Safety and reliability assessment techniques in robotics." Robotica 15.06 (1997): 701-708.