Criterion E

Part 1: Meeting Success Criteria

Success Criteria

- System can detect when there is change in distance from the sensors
- An algorithm can determine whether it is more likely that the sensor reading is a false-positive, a change in ground level, or an actual obstacle to avoid.
- System can determine approximately where the obstacle is located with respect to the robot.
- System can provide relevant information about what it detects so that the main computer on the UGV can determine what actions to take.

The system can detect when there is a change in distance from the sensors. I have developed an algorithm that takes a moving average of the sensors to better avoid false positives, however the sensors still seem to have issues sometimes. Multiple sensors are used to determine approximately where the obstacle is located. The current system does provide information to the main computer on the UGV, but also currently has an algorithm to avoid obstacles when an obstacle in its path is detected. Overall though, the product isn't very successful in a real-world application because of the variation in things such as ground level, and also the amount of sensors means that the project can't be highly accurate with dead spots. My client believes that the main improvement to be had is adding more sensors. Currently with my algorithm that would be impractical simply due to volume of conditionals.

Part 2: Feedback from Client

The client overall thought the product was useful, as evidenced by my final interview with them (see appendix B). They found that having just three sensors wasn't very successful due to the dead spots between each sensor. They also saw that sometimes my algorithm would detect obstacles where there wasn't any.

Part 3: Recommendations for Future Improvements

One improvement that would have been nice to implement is a more clean algorithm. Currently, the algorithm is very hard to read, so any modifications to the code by other computer scientists or programmers may end up making the code function in weird ways. One way the algorithm could be cleaner is by using a state machine so that it is less difficult to keep track of every state the UGV could be in. Another improvement would be to simply add more sensors because that would remove the dead spots between each sensor, which would mean the product would be more successful at obstacle detection. This would be fairly easy to implement due to the use of object-oriented programming, except for having to keep track of which combinations sensors are activated and what to the UGV based on that.