# Engineering Addendum
## Team 24 – UPR Level Security

Many of the setbacks faced by our team were a result of inadequate equipment purchased with insufficient funding. The most important first step for a team attempting to continue this project would be to upgrade the equipment. The Raspberry Pi, used for most of the processing on the robot itself, was not powerful enough to perform the functions required of it. To handle all of the processing that the robot needs to do in real time, a more powerful processor is required (such as the NVIDIA Tegra). If running on the Raspberry Pi, the full multithreaded operation code will completely max out on CPU usage, causing many hard to debug errors. Additionally, the ultrasonic sensors should be upgraded to limit the probability of false detections, which currently happen frequently. Lastly, a more accurate and reliable MPU will be helpful. The current MPU has a tendency to freeze up, which can result in an angle reading being missed (causing the robot to overshoot the desired angle). The angle readings will also occasionally float away from the correct value while the robot is stationary or moving in a straight line. We never determined the cause of this, beyond a cheap MPU. The simplest, and most effective way to improve upon the success of this project will be to upgrade the equipment first and foremost.

One of the largest sources of difficulty with the Web Application came from the deployment to Amazon Web Services. The application is currently designed to use AWS Elastic Beanstalk. If the project were restarted from scratch, it would be advisable to use Amazon EC2 in place of Elastic Beanstalk, due to it being easier to set up. However, given that the current application is set to use Elastic Beanstalk, the decision is not so simple, and it is up to the team working on it to decide which option fits their needs the best.

A major difficulty faced with the implementation of the robot was the communication between the Raspberry Pi and Arduino using the serial port. The serial connection can be read perfectly with the Arduino IDE. When using the main.py code the serial connection was incredibly inconsistent, leading to many missed messages or messages with missing information. We determined that this was mostly a result of the overuse of the CPU. When using a simpler, single-threaded program the serial connection was much more consistent, albeit still not perfect. Any program using the serial connection will need to have some sort of protocol for dealing with errors in the connection.

The GPS position reading for the robot is currently using the raw GPS output, without any RTK corrections. This allows the robot to get roughly 1-2m of accuracy in the very short term, but it will experience drifting leading to accuracy within only 10m over the long term. Using RTK (real time kinematic) corrections should enable the robot to achieve accuracy up to 2cm, which is more than enough. However, these corrections need to come from a base station, which can be difficult and costly to set up. Luckily, the Massachusetts Department of Transportation (MassDOT) offers RTK corrections for free at the following website: http://macors.massdot.state.ma.us/spiderweb/frmIndex.aspx. To receive the corrections, an account must be created and you have to apply for access, which takes a few days. The corrections can only be received through an ntrip connection, and must be decoded using RTCM3. The positionTracker.py file in PiFiles/UPR_Main/RTCM3 provides an example of how to do this. Unfortunately, the data returned from this website does not provide any actual corrections. Corrections are supposed to be returned in RTCM3 1004 and 1012 messages, but there was no useful data returned in these messages from MassDOT, and we could not determine why. The only useful data was returned in the 1006 messages, which gave the current position of the base station, but this was not enough to provide corrections. To be able

to use RTK corrections, this data from MassDOT will need to be turned into some usable form. One thing that we were unable to try, due to lack of time, was the RTKLib library, which should be able to handle corrections for us. However, the RTKLib manual is over 700 pages long and used in many different use cases, so finding the correct use case for this project will take some time.