

### NavX Displacement Data

During the 2015 build season and into the 2016 build season our team has been investigating the use of the navX and the old nav6 sensors. The navX has 3-axis gyros, 3-axis accelerometers, a magnetometer, and a temperature sensor. The navX internally processes these sensor values and sends accurate attitude (yaw, pitch, and roll) data to the roboRIO. The navX also sends velocity and displacement data that is based on the navX's accelerometers to the roboRIO. However, according to Scott Libert the navX displacement data is [only accurate to about a meter every 15 seconds](#)! This was confirmed by our tests. In fact, we often could not tell whether the robot was moving forward or backward by looking at the data from the navX.

After coming to the conclusion that the navX would not be able to tell us the robot's displacement, in any way accurate enough to be useful, we started to develop other methods of navigating the playing field.

### Position Tracking

In order to track the robot's displacement on the playing field in a manner very similar to what we had hoped to get from the navX's accelerometers, we devised a way of using trigonometry to combine the navX yaw angle (which tells us the robot's heading) with the distance the robot has traveled (as measured by averaging the change in distance of each of the drive wheel encoders). In this way the robot's *position* is internally updated many times a second. This works quite well although it only tells us the displacement of the robot relative to where it turned on. Our code is therefore designed to allow the position to be set to any point (not just reset to point 0, 0). This is also the case for the navX yaw so that we can setup the robot on the field in any orientation and the robot will still know its original position and orientation. Additionally, allowing other parts of the code to update the position means the position can be updated by other forms of sensory.

### Traverse XY Coordinates

Once position data is collected, it may be used in any part of the code. The most obvious way it is used is to autonomously traverse the field to get from one position to another and, repeating the process, traverse multiple points. This is accomplished using the following routine (please refer to the power point presentation for pictures). The most important mathematical principle used here is the *slope-intercept formula*.

- Slide 1. When the routine starts, remember the *initial position*. Throughout the routine, the *desired position* and *current position* are known.
- Slide 2. Draw a line from the *initial XY coordinates* to the *desired XY coordinates*. This is the *course* we want the robot to follow.
- Slide 3. Draw another line, perpendicular to the *course*, that measures the robot's *distance off course*.
- Slides 4 - 8. Use the robot's *distance off course* as the process variable in a PID routine and set the setpoint to 0 since we want the robot to gravitate toward 0 feet away from its *course*. This first PID routine calculates the *desired heading* and inputs

this value to a second PID routine as the setpoint. The second PID routine's process variable is the *navX's yaw angle* and the output is used to *rotate* the robot toward its course.

Slide 9. The routine stops the robot and allows automode to move to the next step when the *current position* is on or past a line perpendicular to the *course* and intersecting the *desired position*.

#### Defense Crossed?

The above method works well as long as the robot is on flat carpet and the drive wheels have good traction. (This year our team used six wheel tank drive with pneumatic tires.) This year's field, however, is not all flat carpet! And because our automode plan called for us to go over a defense the first thing, we needed another way of knowing when we had fully crossed the defense. We do this by reading the navX's pitch angle. When the robot is just past halfway over the defense and pitched down (a negative value), the code throws an internal flag. When the robot has almost leveled out again AND the flag has been thrown, the code knows the robot has successfully crossed the defense. At this point, the code internally sets the robot's XY position to a known point (i.e. just beyond the defense) and continues to use the robot's XY position to navigate the field.

#### Record Video and Network Table Variables

The above methods are very useful for approximating our position on the playing field. However, we still wanted to ensure the position was near perfect before approaching the low goal and shooting our one preloaded boulder, so we decided to use vision processing. In order to allow the vision processing to be tuned for, and throughout, every competition we modified the LabVIEW dashboard to automatically record video from the robot's camera during the automode period of each FMS connected match. We also copied the vision processing code to the dashboard so we can calibrate using the video of automode and update the calibration values in the robot code. Besides recording video of automode, we also record the network table variables which contain sensor values and other debugging data from our robot. Being able to review these variables has proved to be very useful for debugging.

#### Cardinal Alignment Buttons

We do not limit ourselves to using the navX yaw strictly in the autonomous mode. We also use it in TeleOp, giving the drivers cardinal alignment buttons. Each of these four buttons is assigned an angle: North (0 degrees), East (90 degrees), South (180 degrees), and West (270 degrees). (When we say the words North, East, South, and West we say North when we really mean *away from the drivers* and East when we mean *to the driver's right hand*, etc. not literal compass directions.) If no Cardinal alignment button is pressed, the driver has full manual control over the robot's rotation. If, on the other hand, one of the Cardinal alignment buttons is pressed, the angle assigned to that button is used as the setpoint for a PID routine whose process variable is the navX yaw, and whose output rotates the robot toward the assigned angle. This is similar to the second PID routine used to drive from one XY position to another. Even after the turn is completed, the driver may continue to hold the button while manually controlling the robot's

fore-aft movement. For example, the north button can be held down while crossing the ramparts, keeping the robot headed relatively straight even when the ramparts attempts to make the robot turn abruptly. The east and west buttons also make driving the robot in low visibility areas much easier. In fact, our drivers do not need a camera to open the drawbridge. Instead, they use either the east or west button to drive behind the drawbridge and then switch to either the north or south button to open the drawbridge.