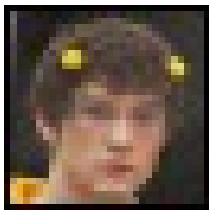
FRC #33 – The Killer Bees
2013 Software – CulverDrive
Algorithm design



VI Icon

Introduction:

The award-winning CulverDrive interface method for skid-steer drivetrains simulates a steering wheel using a gamepad with a round joystick border, allowing the driver fine steering control like with a steering wheel but in the small and handheld package of a gamepad. The Killer Bees used this method successfully through the 2013 season, winning an Innovation in Control award at Waterford and Quality award at the FIRST Championship for the interface method.

History:

Prior to the 2013 FRC season, The Killer Bees used a 'Halo' drive system as for the primary driver controls. The 'Halo' drive system uses the math algorithm from the 'Cheesy' drive (where the left Y controls a 'throttle' input and the right X controls a 'wheel' steering input, and there is a throttle component in steering) with automated switch to 'quick-turn' mode (where the throttle component is removed from the steering, allowing turns in place) when throttle approached zero. We liked it, but we really wanted finer turn control for arc-driving and thought that a steering wheel would provide this fine turn control. However, we did not want to use an actual steering wheel. Bryan Culver (who we would later name the drive system after) suggested using both the X and Y axis of the steering joystick for steering, allowing the user to 'drive' the joystick around the round stick boundary to steer. We implemented the algorithm in C for Vex robots (included in the CulverDrive zip) and later ported it to LabVIEW. Bryan could not come up with a creative name on his own, so I named the system 'CulverDrive' in his honor, and placed his picture in the LabVIEW VI icon.

Calculation of 'Radius' steering component (all numbers are multiplied)
-R of steering stick (sqrt[x^2+y^2])
-Θ normalized of steering stick (0 to 1 from 0 to 90 degrees from vertical)
-Gain (static calibrated number)
-Throttle
-Enablement curve (1 from |0| to |95| degrees, fades to 0 by |115| degrees)

Calculation of 'Raw' steering component:
-R of steering stick
-Θ normalized of steering stick
-Throttle
-Gain
-Enablement curve of Radius

Alternate calculation of 'Raw' steering component:
-R of steering stick
-Sign of Θ
-Gain
-Enablement curve (0 from |0| to |90| degrees, 1 from |135| to |175|, 0 at |180|)

The switch from Radius to Raw steering mode is done with a 'Quick-Turn' button, similar to the Cheesy Drive. We experimented with a method where the sum of Radius and Alternate Raw were simply added together (their enablement curves allowed them to fade between each other) but did not like it.

The image above shows the use of a Logitech F310 joystick with the CulverDrive. The left Y axis is used as the 'throttle' input (as with the Cheesy and Halo drives), but the right Θ*r is used as the primary component of the steering input instead of the right X as a Cheesy or Halo drive. The turn power is then calculated similar to a Cheesy drive, where in Normal/'Speed-Turn' mode the steering input (Θ*r) is multiplied by the throttle to provide good arc-turn performance at any speed, and the steering input is used directly when in 'Quick-Turn' mode to allow turning in place at zero throttle.

Layout and architecture of example code:
The example code was written to run in simulation and eventually a Vex target. Unfortunately, RobotC does not support typedef structs as function return types, and seems to have issues with floats in function calls, so the code will not compile in RobotC.

The example code in C contains five files:
-culverdrive.h contains the function prototype and externs
-culverdrivedata.c contains const and cal data definitions
-culverdrive.c contains the cheesy drive algorithm code. The code is a single function with void return, data is returned via pointers to left and right wheel powers.
-intrerp.h contains the header for the interpolation blocks. In LabVIEW these are part of BuzzLib. The code is functionally identical to the LabVIEW implementation.
-interp.c contains the code for the interpolation blocks.