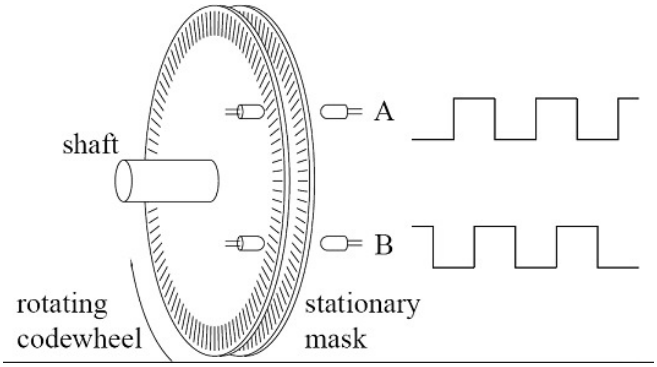
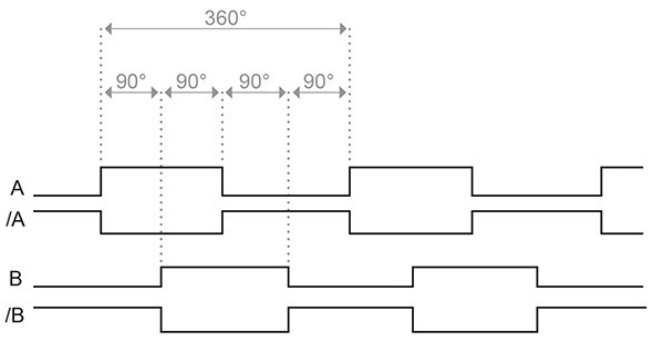
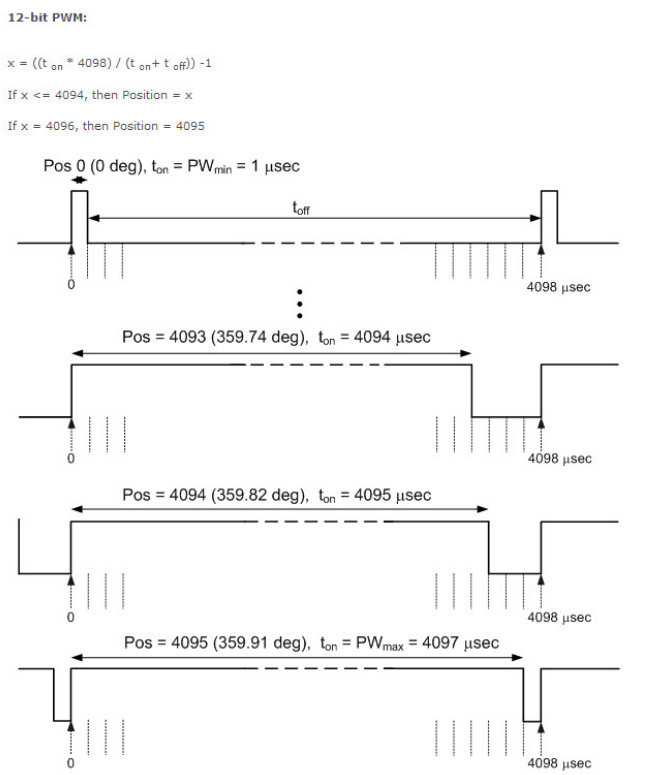
How to take down Class C Defences
by team 4678, Woodland CyberCavs

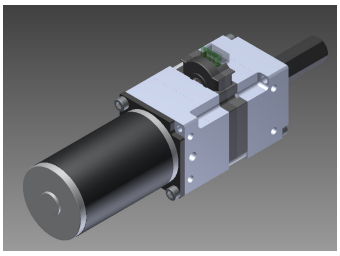
Step 1. Understand how incredibly useful the Vex Versaplanetary Integrated Encoder really is when you plug it into a Talon SRX. This encoder is not just a quadrature encoder. It provides both a quadrature signal and an absolute position signal. This means you can know exactly what position your appendage is in without having to home it first. You just have to make sure you use it right (which we didn't exactly do by the way).

In order to be properly educated about encoders, you need to know the difference between an incremental (quadrature) encoder and an absolute encoder. Here's a chart that has the potential to explain the differences ...

Incremental (Quadrature)	Absolute (PWM)
 	
Position is calculated very, very quickly, essentially no delay so you could, in theory, determine the precise position of something spinning at 10,000 rpm or more.	Position can only be updated as often as it takes for the PWM cycle (about 4100uS or about 243Hz for the above diagram)
Java example, Elbow.getEncPosition()	Java example Elbow.getPulseWidthPosition()

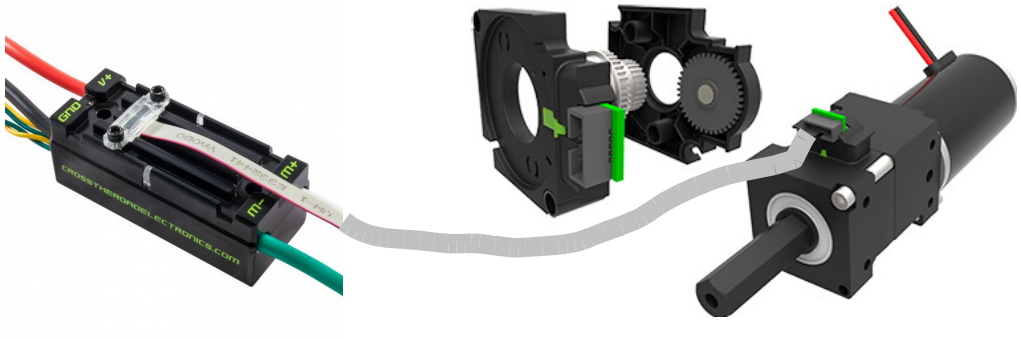
The VersaPlanetary Integrated Encoder actually uses a single magnet and some cool hall-effect sensor technology to determine the exact angular position of the magnet. Then it simulates the incremental quadrature output and generates the PWM output to report the absolute position.

Step 1 continued...
So, where, in a Versaplanetary Gear Assembly, do you put the encoder? It could go in 1 of 3 places on a 2-stage gearbox. This drawing shows it in the middle position between the 2 gear reduction stages. This is where we mounted it on the 2 drive motors for our manipulator arm and it wasn't the right spot.



At first, we had it right next to the motor. This would give us maximum accuracy, we thought. It sure gave us high resolution. Our arm positions measured from 0 to about 500,000. When we moved the encoder to the central position between the gear reduction states, we were dealing with positions in the 0 to 50,000 range (makes sense since we were using 2 reductions stages of 10:1 each). This is still way more resolution than you really need. The correct place to put the encoder is normally going to be in the last position after the reduction stages and just before the output shaft stage. This will give you 4096 counts of angular position resolution which should be plenty good enough for almost all motion control situations on an FRC robot.

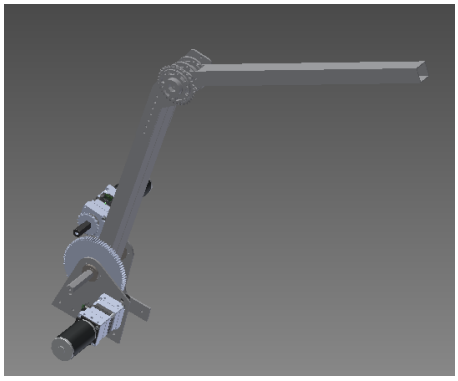
The problem with placing the encoder right by the motor is that the start-up position (when we turn on our robot) can easily be different by 1/100 of a full rotation which then means the magnet might be rotated + or – 360° and the absolute encoder can only tell you your position within 360° when you first turn on your robot. Even when the encoder is between the 2 reduction stages, we still had problems with accurate arm positioning if we happened to boot up our robot when the arm was not properly seated in its start-up position. It would have been far better to have placed the encoder right next to the output stage on the versaplanetary gearbox. This would allow the robot to read an accurate angular arm position even if we didn't have it in quite the right spot when we started up the robot.



In our code, one of the first things we would do at start-up is read the Absolute Encoder position and set the incremental position to match it.

```
wristStartPosition = (manipulatorWrist.getPulseWidthPosition() % 4096);
elbowStartPosition = (manipulatorElbow.getPulseWidthPosition() % 4096);
if (wristStartPosition < 0) {
      wristStartPosition = wristStartPosition + 4096;
}
if (elbowStartPosition < 0) {
      elbowStartPosition = elbowStartPosition + 4096;
}
manipulatorWrist.setPulseWidthPosition(wristStartPosition);
manipulatorElbow.setPulseWidthPosition(elbowStartPosition);
manipulatorWrist.setEncPosition(wristStartPosition);
manipulatorElbow.setEncPosition(elbowStartPosition);
```

Step 2: Build an arm with 2 degrees of freedom like this one. Elbow powered by a bag motor with a 100:1 Versaplanetary further geared down with 20 Tooth to 84 Tooth set of hex gears. Wrist powered by a bag motor with 100:1 Versaplanetary and a set of sprockets and chain so the motor could be mounted lower down on the arm. Both gearboxes equipped with the VersaPlanetary Encoder and Talon SRX motor controllers.



Step 3: Gain an understanding of some of the potential traps that exist in the Roborio operating system and talon SRX firmware.

We were originally informed that the execute() function for a command in java would be processed 50 times per second. If you really look into the documentation that states this, you will find that it says “normally” 50 times per second. It turns out that wireless network communication glitches and any number of other potential hiccups between the driver station and the roborio are conveniently solved by skipping the execute() that would normally take place for that cycle (and sometimes the next 2 or 3 as well). This is a big problem when you want to make the robot move something in a very controlled and consistent manner. To account for a rather unpredictable rate of execution, our new friend is the getFPGATime() function. This reads an integer that counts at exactly 1,000,000 counts per second. We can use getFPGATime() and some calculations to figure out exactly where we want our wrist and elbow positions to be at that precise moment and simply send that position information to the Talon SRX. In our code, we found it handy to use the floating point version Timer.getFPGATimestamp(); which reports the time in seconds.

```
import edu.wpi.first.wpilibj.Timer;
fpgaDiff = Timer.getFPGATimestamp() - lastFPGA; //accurate time tracker
lastFPGA = Timer.getFPGATimestamp();
```

The other interesting thing we stumbled across is that the Talon SRX keeps track of position information internally in real time but only updates the position information it sends to the roborio every 1/10th of a second. Sometimes we would read the motor position values and we'd get the same value 5 times in a row (when execute() actually was running at 50 times per second). Turns out there is a command to have the SRX update the position information at more like 100 times per second.

```
import edu.wpi.first.wpilibj.CANTalon.StatusFrameRate;
manipulatorElbow.setStatusFrameRateMs(StatusFrameRate.QuadEncoder, 10);
```

This sets the update rate to 10ms instead of the default 100ms.

Step 4. Use the encoders on the manipulator arm and the drive train to take readings of several positions that need to be achieved in order to open the drawbridge using the robot as the tool to do the opening. For our robot, this required only about 5 distinct positions to get the drawbridge fully open. Normally, the robot starts up with the arm axes and the drive train in a “disabled” state so we can manually move them without a problem when the robot is powered up and enabled. By placing the encoder values on the smart dash board, we were able to record the encoder readings of the 2 arm axes and the 2 drive train encoders at the 5

different positions. We then built our program code to have the robot move the 2 axes and the drive train to reach these target positions with all the motors moving simultaneously.

Because there were only 5 sets of values, we hard-coded the motion to move from one position to the next using case statements. This allows special considerations to be implemented rather than just flat out “position to target” of each axis. What follows is our drawbridge code. This has been modified slightly from the code found in our github repository *https://github.com/Woodland4678/Cybercavs2016Code/releases/tag/V1.0* in that extra comments have been added to try to explain things more clearly.

```
public boolean drawBridge() {
     fpgaDiff = Timer.getFPGATimestamp() - lastFPGA; // Use getFPGATime to avoid problems with execute() being skipped occasionally.
     lastFPGA = Timer.getFPGATimestamp();           // Find the difference from the last time this was executed and update last.
     System.out.println("ACTUAL TIME: " + Timer.getFPGATimestamp());
     switch(drawBridgeState) {
     case 0: // Moves the wrist first so that the arm doesn't scrape against the drawbridge
          Robot.robotDrive.resetEncoders();                    // Set the drive train position to 0.
          setManipulatorWrist(drawBridgeWristReady);           // Send the position command to the wrist axis.
          if (manipulatorWrist.getError() < 1000 && count > 10) { // this determines if the wrist is in position,
                                                               // if so it moves the elbow
               drawBridgeState++;                              // Wrist is in position, go to next state to move the elbow
               count = 0;
          }
          count++;
          break;
     case 1: //moves the elbow until its in position, then moves on to next state
          setManipulatorElbow(drawBridgeElbowReady);           // Send the Elbow to first position for drawbridge crossing.
          if (manipulatorElbow.getError() < 200 && count > 10) { // When we're close enough to target position, go to next step.
               drawBridgeState++;
               count = 0;
          }
          count++;
          break;
     case 2: //sets the wrist to power mode and gives is 10% power downwards to apply some pressure while opening the drawbridge.
          manipulatorWrist.changeControlMode(TalonControlMode.PercentVbus); //power mode (values between -1 and 1)
          manipulatorWrist.set(0.1);                           // Set value to 10%.  This is enough to move the wrist down
          drawBridgeState++;                                   // with just a bit of pressure.
          timeCount = 0;                                       // reference time is set to 0
          totalTime = 0.75;                                    //this indicates how long the next step has to finish
                                                               // a lower value will make it go faster with less accuracy.
          break;
     case 3: //calculates the next elbow and drivetrain position based on time, updates speed every 50th of a second
             // This step pulls bridge half way down, and drives back a little.
          timeCount += fpgaDiff; //keeps track of time
          // Calculate the new elbow position by interpolating the position we should be at given the value in timeCount
          // by the time timeCount is about equal to totalTime, we should be at our target.
          newElbowPosition = (elbowSecondPosition - drawBridgeElbowReady) * timeCount / totalTime + drawBridgeElbowReady;
          setManipulatorElbow(newElbowPosition);               //set manipulator elbow to new position
          // Calculate where the drive train should be.  The robot should be moving away from the drawbridge at the same
          // time as the elbow is pulling the drawbridge down.
          encoderPosition = encoderSecondPosition * timeCount / totalTime;
          speed = (encoderSecondPosition / totalTime) / SPEED_FACTOR; //speed of drivetrain
          // DrivePath will actively attempt to get the drive motor encoders to match the value in encoderPosition
          // This is a PID function.  speed helps set the power level so the routine doesn't need to work with just position.
          Robot.robotDrive.DrivePath(encoderPosition, speed, encoderPosition, speed); //makes robot drive on specified path.

          if (timeCount > totalTime) { //if time exceeds that specified in previous state it moves on to the next and sets a time
                                  // limit for the next step
               manipulatorWrist.changeControlMode(TalonControlMode.Position); //changes wrist to position mode
               setManipulatorWrist(wristSecondPosition); //makes wrist maintain current position
               timeCount = 0;
               totalTime = 0.75;                       // Prepare for the next movement transition. Allow 0.75 seconds.
               drawBridgeState++;
          }
          if (manipulatorWrist.get() > wristSecondPosition - 500) { //checks if manipulator reaches next target before this step
                                                               // finishes and it then maintains that position
               manipulatorWrist.changeControlMode(TalonControlMode.Position);
               setManipulatorWrist(wristSecondPosition);
          }
          break;
     case 4: //pushes arm forward and downwards to push drawbridge fully down, drivetrain moves back more so drawbridge doesn't
                                                                                    // hit the bumper.
          timeCount += fpgaDiff;
          newElbowPosition = (elbowThirdPosition - elbowSecondPosition) * timeCount / totalTime + elbowSecondPosition;
          newWristPosition = (wristThirdPosition - wristSecondPosition) * timeCount / totalTime + wristSecondPosition;
          encoderPosition = (encoderThirdPosition - encoderSecondPosition) * timeCount / totalTime + encoderSecondPosition;
          speed = ((encoderThirdPosition - encoderSecondPosition) / totalTime) / SPEED_FACTOR;
          setManipulatorElbow(newElbowPosition);
          setManipulatorWrist(newWristPosition);
          // By the time the robot has backed up to the target position, the robot frame will be back far enough that the
          // drawbridge will not catch on it as we push the drawbridge fully open.
          Robot.robotDrive.DrivePath(encoderPosition, speed, encoderPosition, speed);
          if (timeCount > totalTime) {
               drawBridgeState++;
               timeCount = 0;
               totalTime = 0.5; //next step only gets 1/2 second to finish
          }
          break;
     case 5: //pushes arm out a bit more (both wrist and elbow move) to keep drawbridge down, and begins to drive forward
```

```
                timeCount += fpgaDiff;
                newElbowPosition = (elbowFourthPosition - elbowThirdPosition) * timeCount / totalTime + elbowThirdPosition;
                newWristPosition = (wristFourthPosition - wristThirdPosition) * timeCount / totalTime + wristThirdPosition;
                encoderPosition = (encoderFourthPosition - encoderThirdPosition) * timeCount / totalTime + encoderThirdPosition;
                speed = ((encoderFourthPosition - encoderThirdPosition) / totalTime) / SPEED_FACTOR;
                setManipulatorElbow(newElbowPosition);
                setManipulatorWrist(newWristPosition);
                Robot.robotDrive.DrivePath(encoderPosition, speed, encoderPosition, speed); // Move forward to get robot wheels on the
                                                                                            // drawbridge.
                if (timeCount > totalTime) {
                     drawBridgeState++;
                     timeCount = 0;
                     totalTime = 0.75;
                     manipulatorWrist.changeControlMode(TalonControlMode.PercentVbus);
                     manipulatorWrist.set(0);
                }
                break;
           case 6: //drive forward the rest of the way.
                timeCount += fpgaDiff;
                newElbowPosition = (elbowFifthPosition - elbowFourthPosition) * timeCount / totalTime + elbowFourthPosition;
                encoderPosition = (encoderFifthPosition - encoderFourthPosition) * timeCount / totalTime + encoderFourthPosition;
                speed = ((encoderFifthPosition - encoderFourthPosition) / totalTime) / SPEED_FACTOR;
                setManipulatorElbow(newElbowPosition);
                Robot.robotDrive.DrivePath(encoderPosition, speed, encoderPosition, speed);

                if (timeCount > totalTime) {
                     drawBridgeState++;
                     timeCount = 0;
                     totalTime = 0.3;
                }
                break;
           case 7: //moves elbow up a little bit so the manipulator arm doesn't get stuck under the robot
                timeCount += fpgaDiff;
                newElbowPosition = (elbowSixthPosition - elbowFifthPosition) * timeCount / totalTime + elbowFifthPosition;
                setManipulatorElbow(newElbowPosition);
                Robot.robotDrive.DrivePath(encoderFifthPosition, 0, encoderFifthPosition, 0); //stop
                if (timeCount > totalTime) {
                     drawBridgeState++;
                     timeCount = 0;
                     totalTime = 1.0;
                }
                break;
           case 8: //stops the drivetrain so it doesn't continue at previous speed
                manipulatorWrist.changeControlMode(TalonControlMode.Position);
                Robot.robotDrive.DrivePath(encoderFifthPosition, 0, encoderFifthPosition, 0); //stop
                return true;
           }
           System.out.println("Case: " + drawBridgeState + ", " + "Angular wrist: " + Robot.manipulatorArm.getManipulatorWristAngular() +
", " + "Wrist encoder: " + Robot.manipulatorArm.getManipulatorWristPosition());
           return false;
      }
```

The complete source code for our robot can be found on our github release.
https://github.com/Woodland4678/Cybercavs2016Code/releases/tag/V1.0