# THE NERD HERD

## FRC TEAM 687

## SOFTWARE AND CONTROLS

# Table of Contents

# Robot Overview

- 6-Wheel West Coast Drive
  - 2 speed
  - 4 omni wheels and 2 traction wheels provide reduced turning scrub
  - Powered by 6 Mini CIM motors
  - SRX Mag Encoders on each side
  - NavX IMU
- Cascading Elevator
  - 2 stages
  - VersaPlanetary Integrated Encoder attached to end of gearbox
- Virtual Four Bar
  - Mounted on the elevator carriage
  - VersaPlanetary Integrated Encoder attached to end of gearbox
- Intake
  - Mounted on virtual four bar
  - 2 side rollers for intaking Cargo
  - Pneumatically actuated to close and intake Hatch Panels

# Programming Team

- Program mainly in Java
  - Vision processing is done in Python on JeVois
- 19 Programmers
  - 5 rookies
  - 11 veterans
  - 3 leads
  - Large programming team allows us to explore multiple solutions to problems
- Subteam splits into several groups over the course of the season
  - Autonomous
  - Vision (JeVois)
  - Vision (Limelight)
  - Line tracking
  - Superstructure/scoring mechanisms
  - Miscellaneous sensor testing (hall effect sensors, ultrasonics)
- As various tasks approach completion, it's common for our programmers to shift around between projects
- All code is uploaded to Github at the end of meetings
  - https://github.com/nerdherd
- Whole subteam gives an update on what they've done on Slack after the meeting
  - Keeps everyone "in the loop"

# Code Structure

- WPILib Command Based Structure
- Subsystems
  - One set of actuators that function in unison
  - Drivetrain, intake, elevator, etc
- Commands
  - Actions for an subsystem to perform
  - Move to a position
  - Drive an auto path
- Logging
  - Every subsystem contains methods to log sensor data to CSVs on a flash drive mounted on the roboRIO
  - Logging uses a modified version of FRC Team 1014's Badlog logging system
  - Logging allows us to troubleshoot problems
  - Lets us empirically measure feedforwards for more robust control loops
  - SFTP into the roboRIO using Filezilla to retrieve logs
  - Open and analyze logs using Logger Pro

# NerdyLib

- Repository for year agnostic code
- Can be accessed as a Gradle dependency via Jitpack
- Makes use of FRC Team 449's dependency-injection model
  - Generic subsystems can easily be created, and "injected" into generic commands requiring them
- Allows less code to be written each year
- New code only has to be written to integrate mechanisms together
  - Each subsystem is effectively just a collection of configuration methods for the generic form of the subsystem

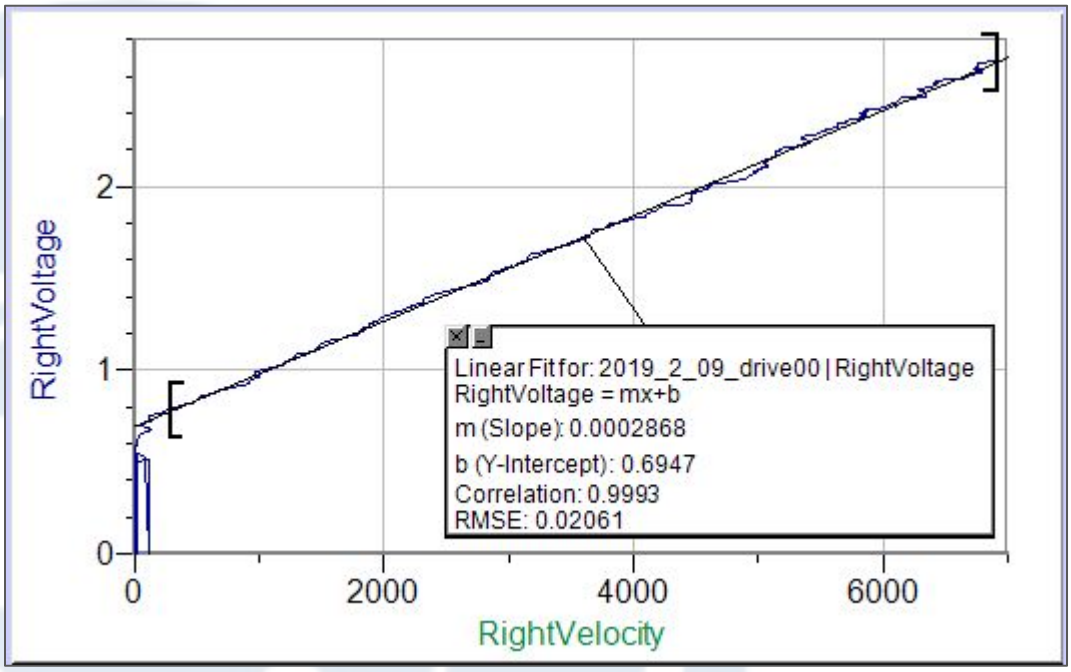
```
public Drive() {
  super(RobotMap.kLeftMasterTalonID, RobotMap.kRightMasterTalonID,
   new VictorSPX[] {
     new VictorSPX(RobotMap.kLeftSlaveVictor1ID),
     new VictorSPX(RobotMap.kLeftSlaveVictor2ID)
   },
   new VictorSPX[] {
     new VictorSPX(RobotMap.kRightSlaveVictor1ID),
     new VictorSPX(RobotMap.kRightSlaveVictor2ID)
   },
   true, false,
   new Piston(RobotMap.kDrivetrainShifter1ID, RobotMap.kDrivetrainShifter2ID));

   super.configAutoChooser(Robot.chooser);
   super.configMaxVelocity(DriveConstants.kMaxVelocity);
   super.configSensorPhase(false, true);

   super.configTicksPerFoot(DriveConstants.kLeftTicksPerFoot, DriveConstants.kRightTicksPerFoot)
   super.configDate("2019_2_15_");
   // floor
   super.configLeftPIDF(0.0, 0, 0, DriveConstants.kLeftF);
   super.configRightPIDF(0.0, 0, 0, DriveConstants.kRightF);
   super.configStaticFeedforward(DriveConstants.kLeftStatic, DriveConstants.kRightStatic);
}
```

# Teleoperated Control

- 3 Logitech Attack 3 Joysticks
  - 2 for driver
  - 1 for operator
- Standard 2 joystick arcade drive
  - Buttons for high gear and low gear
- Button for enabling vision driver assist
  - Uses JeVois and vision tracking to steer the robot towards a vision target
  - Linear speed of the robot is still controlled by the driver
- Buttons for high, middle, and low rocket positions
  - Operator can switch between Hatch and Cargo modes
- Buttons for outtaking, intaking, and stopping intake rollers
- Buttons for opening and closing the intake
- Button for beginning climbing sequence
- Camera feed appears on SmartDashboard
- Information operators need appears on SmartDashboard (eg is there a vision target, is the robot in hatch or cargo mode, etc)

# Mechanism Characterization

- Our virtual four bar, elevator, climber arms, and drivetrain are all characterized using a process similar to that in FRC 449's drivetrain characterization paper
- Voltage and velocity have an almost linear relationship
- We can open-loop at a shallow ramp rate to get the voltage vs velocity line
- Y-intercept of the line is the static friction and/or gravity feedforward
- Slope of the line after being scaled is the velocity feedforward




Drivetrain Right Side Voltage vs Velocity Graph

# Scoring Mechanisms

- All scoring mechanisms:
  - Static friction feedforward added when not moving (speed below a certain minimum value)
  - CTRE TalonSRX Motion Magic feature uses velocity feedforwards and a position PID controller to follow a trajectory
    - TalonSRX runs control loops at higher frequencies than the roboRIO for better robustness and responsiveness
    - Accelerates to maximum velocity, cruises, and then decelerates to reach a setpoint (trapezoidal motion profile)
    - PID tuned heuristically to achieve fast step response time
- Elevator:
  - Constant voltage feedforward to compensate for gravity, added to main control loop
  - ~1.5 seconds to reach full elevator height
  - Conversions between raw sensor readings and elevator height in inches
- Virtual four bar:
  - Voltage feedforward varying with cosine of arm angle, added to main control loop
  - ~1 second to travel full range of arm at lowest elevator position
  - Conversions between raw sensor readings and four bar angles

# Path Following

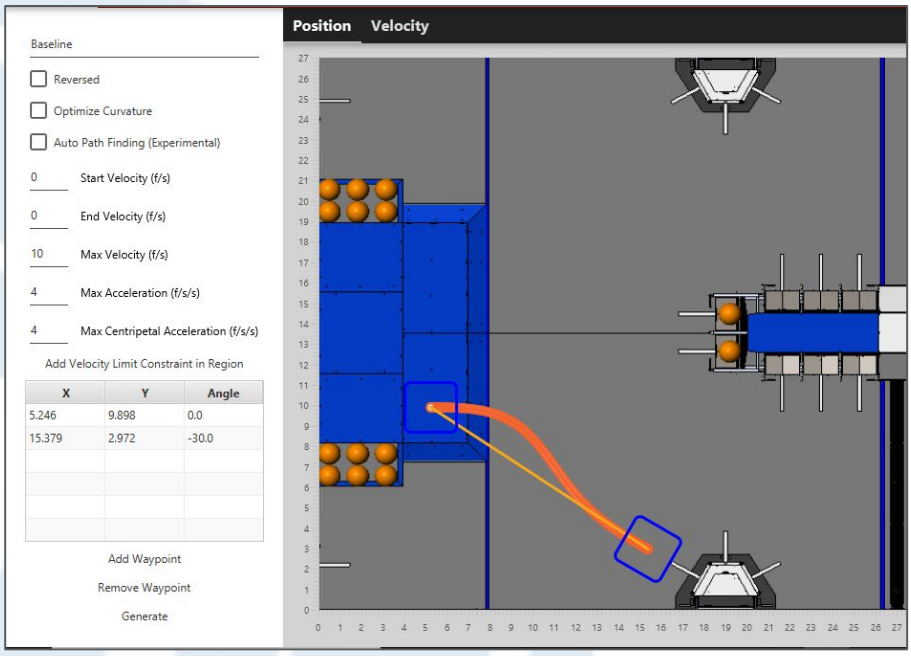
- Trajectory:
  - A quintic hermite spline, time interpolated with velocity, acceleration, position, and angle at every point on the line
- Odometry
  - Keeping track of the robot's x,y position
  - Uses change in encoder position between timesteps and robot's angle
- Generate trajectories using FRC Team 5190's FalconLibrary
- Follow trajectories using a custom Proportional-Derivative Heading controller
  - PD controller is relatively simple
  - Odometry allows us to keep track of our position and correct
  - Trajectory Follower "chases" a point a few timesteps ahead of the robot's current position
  - Controller outputs a left and right velocity in feet per seconds
  - Drivetrain class converts ft/s to Talon velocity units
  - Onboard TalonSRX velocity PIDF is used for velocity control
  - Velocity PIDF can be tuned quickly using drivetrain characterization

```
public void calcXY() {
        double m_currentDistance = (getRightPositionFeet() + getLeftPositionFeet()) / 2;
        double m_distanceTraveled = (m_currentDistance - m_previousDistance);
        double angle = getRawYaw();
        m_currentX = m_currentX + m_distanceTraveled * Math.cos(Math.toRadians(angle));
        m_currentY = m_currentY + m_distanceTraveled * Math.sin(Math.toRadians(angle));
        m_previousDistance = m_currentDistance;
}
```

Odometry

# Autonomous

- Auto modes
  - Right side rocket auto
  - Left side rocket auto
- Robot follows a trajectory until it's close to the vision target
- Vision tracking takes over and steers the robot to the target
- Ultrasonic sensor measurements are used to determine when the robot is close enough to outtake
- Drivers have a button to press to take over control at any moment




Right Rocket Path
(Generated using FRC 5190's Falcon Dashboard)

# JeVois Smart Camera

- Using JeVois Smart Camera for Destination: Deep Space in combination with green LED rings

- Advantages
  - Inexpensive ($60 kit)
  - GRIP compatible
  - Connects to roboRIO through USB or TTL
  - Video stream (30 fps) and data stream simultaneously with USB

- Disadvantages
  - Requires some Python and OpenCV knowledge
  - Initial setup and tuning is tedious

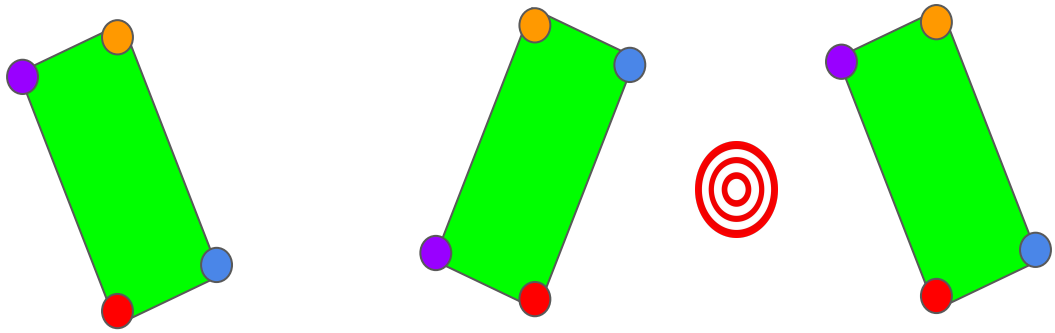
- Custom 3D printed mount
  - Angled 10 degrees inwards
  - Ring lights attached via zip ties
  - Riveted onto the side of the elevator

# GRIP Pipeline

- GRIP utilizes OpenCV functions and an interactive UI to create vision processing systems
  - Blur
    - Minimized blur from a range of 0-2; otherwise the target loses its rectangular structure
  - HSV Threshold - filters image for only green contours
    - H range: 57-91
      S range: 181-225
      V range: 74-173
  - CV Erode - removes noise outside of contour
    - Iteration: 2
  - CV Dilate - repairs damage done with erode; takes contours and enlarges them
    - Iteration: 4
  - Convex Hulls - processes the contour; establishes contour boundary
  - Filter Contours - Filters out contours given constraints Settings are set at default except for Minimum Area
    - Minimum Area: 150 - change based on desired distance to detect target

# Python Module

- Modified JeVois Python module for Destination: Deep Space



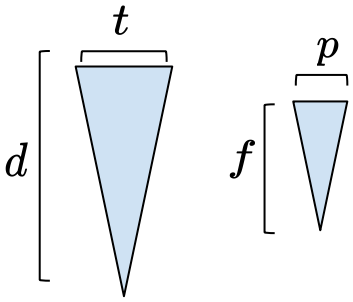
- Differentiate between “hills” and “valleys”
  - “Hills” are actual targets (right two strips)
  - “Valleys” are upside down targets (left two strips)
  - `cv2.boxPoints` returns the corner points beginning with the bottom most point then rotates clockwise. This example returns (x,y) of **RED, PURPLE, ORANGE, BLUE**
  - If **(y_blue - y_red) > (y_purple - y_red)**
    the contour is **tilted right**
    If **(y_blue - y_red) < (y_purple - y_red)**
    the contour is **tilted left**
    Otherwise the contour is vertical
  - Check that **tilted left** and **tilted right** exist, and that **x_tilted right < x_tilted left**
  - Find the center of the two contours
- JeVois module returns number of contours detected, total area, x_center, y_center, height of target, width of target, and distance to target

# Calculations

$$f = focal\ length$$
$$d = distance$$
$$t = target\ width_{inches}\ (constant)$$
$$p = target\ width_{pixels}$$

$$f = \frac{width_{img}}{2 * \tan(FOV \div 2)}$$

$$\frac{f}{p} = \frac{d}{t} \rightarrow d = t * \frac{f}{p}$$




- Angle to target
  - Accounts for horizontal offset from center and horizontal angle offset

```
public double getAngleToTurn() {
        double radians = (Math.PI / 180) * (xPixelToDegree(getTargetX()) + VisionConstants.kCameraHorizontalMountAngle);
        double horizontalAngle = Math.PI / 2 - radians;
        double distance = getDistance();
        double f = Math.sqrt(distance * distance + Math.pow(VisionConstants.kCameraHorizontalOffset, 2)
                        - 2 * distance * VisionConstants.kCameraHorizontalOffset * Math.cos(horizontalAngle));
        double c = Math.asin(VisionConstants.kCameraHorizontalOffset * Math.sin(horizontalAngle) / f);
        double b = Math.PI - horizontalAngle - c;
        double calculatedAngle = (180 / Math.PI) * (Math.PI / 2 - b);
        if (getTargetX() == 0) {
                return 0;
        } else {
                return calculatedAngle;
        }
}
```

- Able to apply P controller to distance and angle for autonomous or teleoperated commands