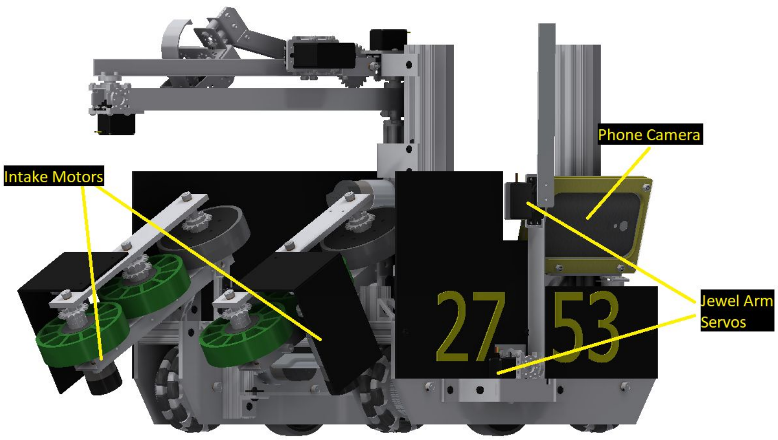
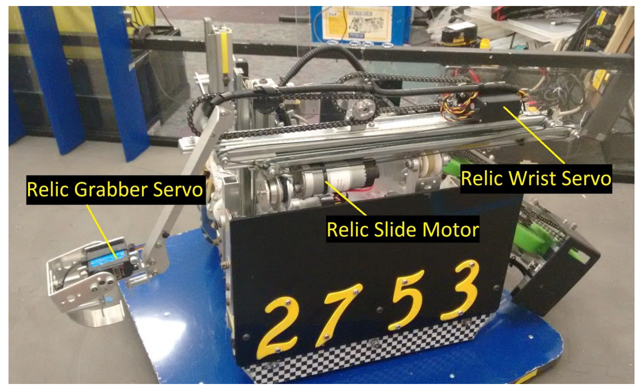
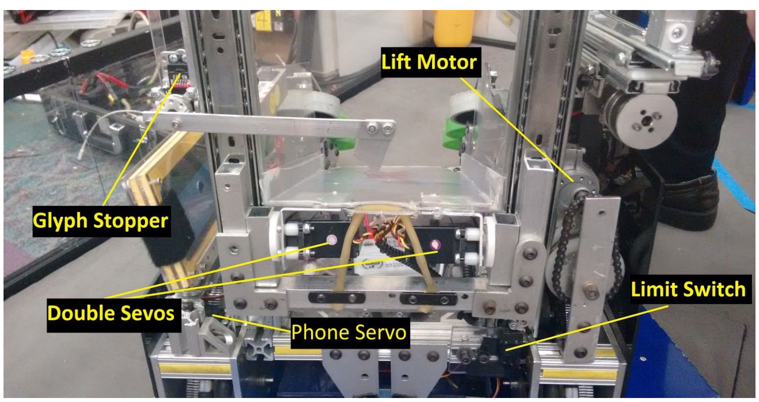
Maryland Tech Invitational 2018
Innovate Sensor Fusion Award Submission

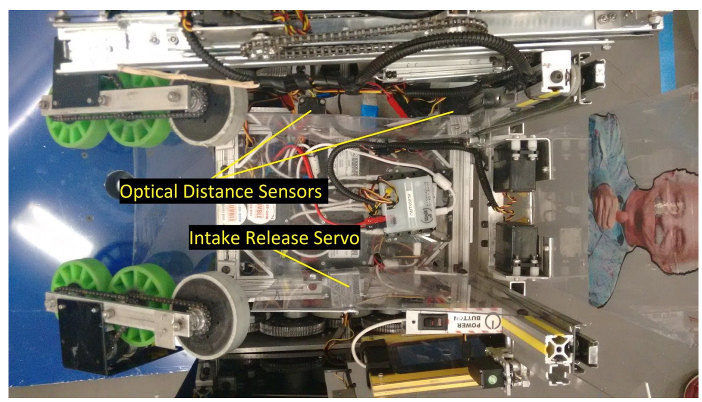
Team Number: 2753
Team Name: Team Overdrive

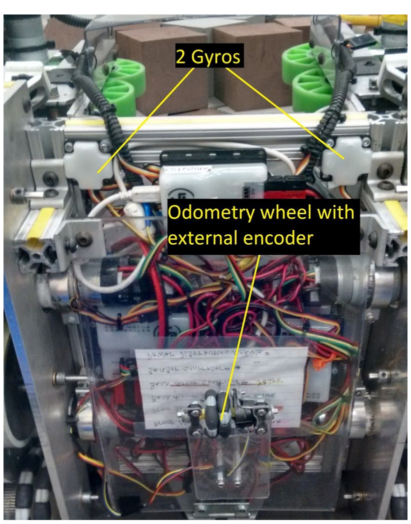
Physical Robot Features

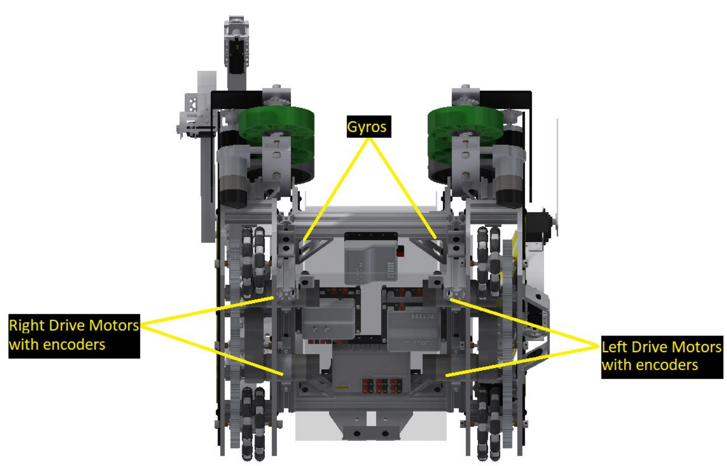












Physical Features

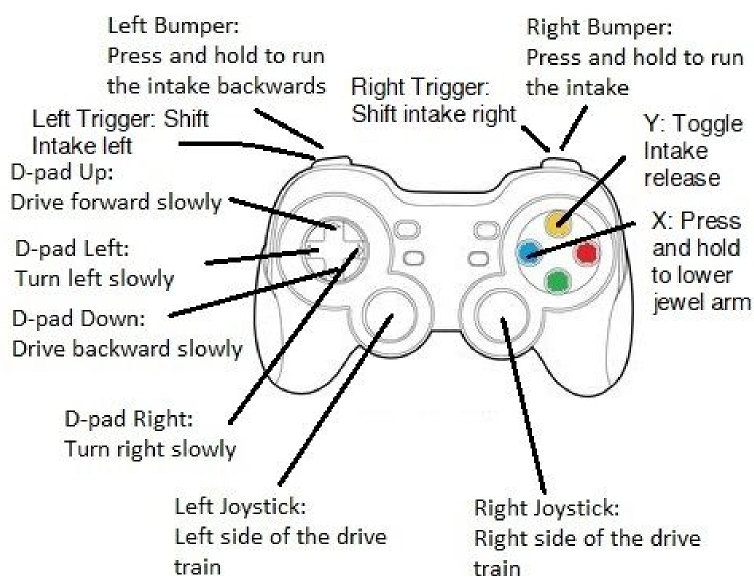
Sensors are denoted in **Red**.

- Drivetrain Module
  - Drive Motors with **encoders**
  - **External odometer wheel**
    - Used in conjunction with the drive train motor encoders
    - Provides a fallback sensor if the motor encoders on the drivetrain fail.
  - **2 Gyros** - used to determine the angle of the robot
    - We have found that 2 gyros working in conjunction gave us more accurate results, less drift, and less failure.
- Intake Module
  - Intake motors
  - Intake release servo
- Lift Module - lifts and places glyphs
  - Lift motor with **encoder**
  - Used to automate glyph placing process in teleop.
  - Double servos to place glyphs
    - Uses a Discrete State-Based Controller to make sure the lift, “slammer”, and “stopper” do not collide during operation.
  - **Limit switch**
    - Used to determine if the lift slides are fully lowered.
    - Used to automate glyph placing process in teleop.
  - **2 Optical distance sensors**
    - Detects the presence and color of glyphs in the glyph tray.
    - Used to automate glyph placing process in autonomous.
- Relic module - used to grab and place relic
  - Motor to extend and retract
  - Servo to actuate relic grabber assembly
  - Servo to grab relic
  - Swings out when intake is deployed
- Jewel Arm - knocks jewel from jewel stand
  - 2 servos for 2 degrees of freedom
- Phone Module
  - **Phone Camera** to detect VuMarks and jewel at the same time
  - Phone servo to rotate phone

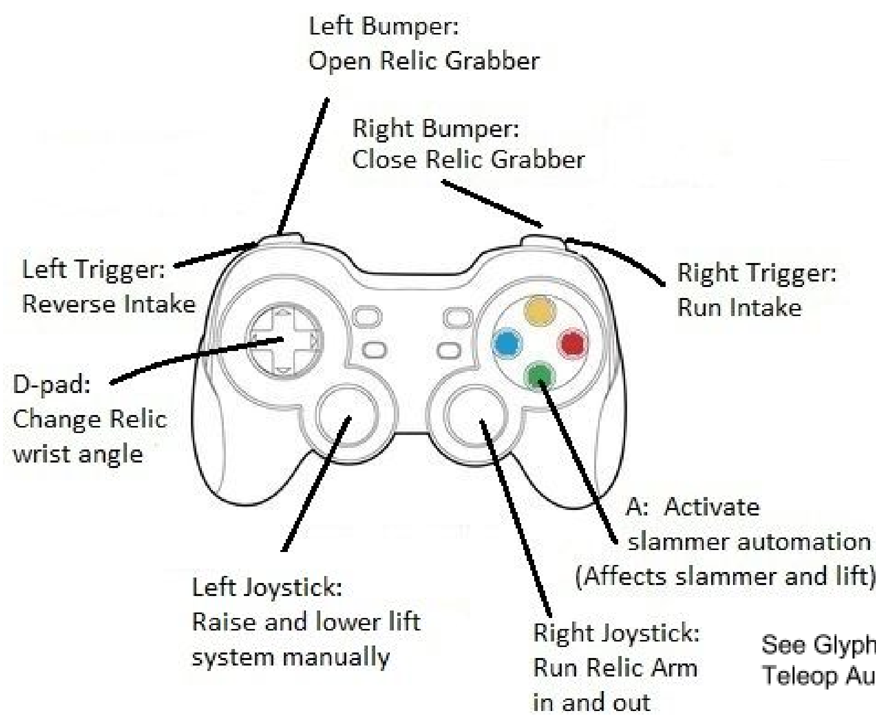
Teleoperated Mode

Controller Diagram

Gamepad 1 Controls



Gamepad 2 Controls



See Glyph Slammer Teleop Automation below

Teleop Logic

- During teleop we automated subsystems that would allow the driver to control the mechanism easily.

  Lift and slammer automation
  - Uses the lift motor **encoder** and the lift slide **limit switch**.
  - This automation helps our drivers place glyph into the cryptobox efficiently.
  - The slides can still be manually lifted and lowered with a controller joystick.
  - The algorithm uses a discrete state-base controller with the controller button and limit switch as conditions and the resulting situations as states.
  - There are three states: Intaking, Lifting/Holding, and Scoring
  - Figure 1-1 shows this process.

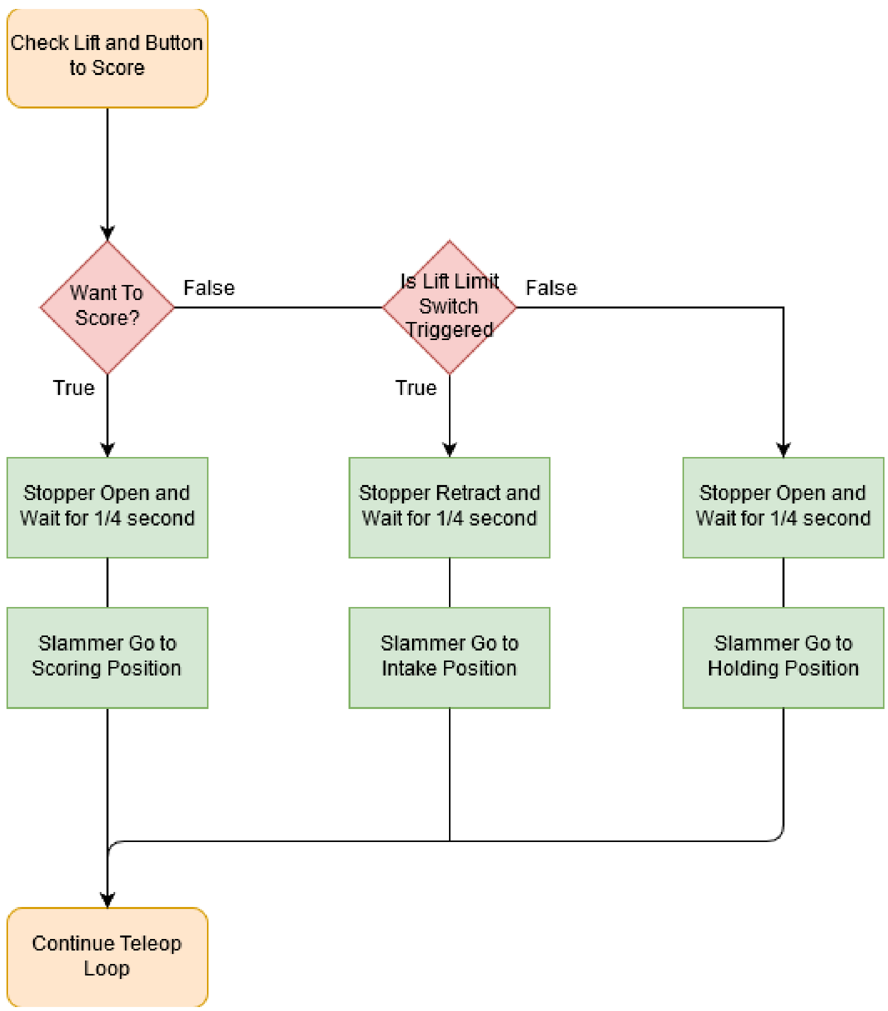


Figure 1-1

Autonomous Mode

Autonomous Algorithms

Path Generation

- Generate a Cubic Hermite spline for the robot to follow (See *Figure 1-2)*, based on user defined points (x, y, θ theta). This spline is then converted into data that the robot can follow. We use a Sinusoidal Curve for all splines. We can also generate Trapezoidal Curve for simpler movements. See *Figure 1-3* the trajectory for the **blue** line (Right Wheel Trajectory). Use cases highlighted in **orange**.
- Used so we can go to any position on the field at the maximum allowable speed of the robot, while maintaining a high degree of accuracy.

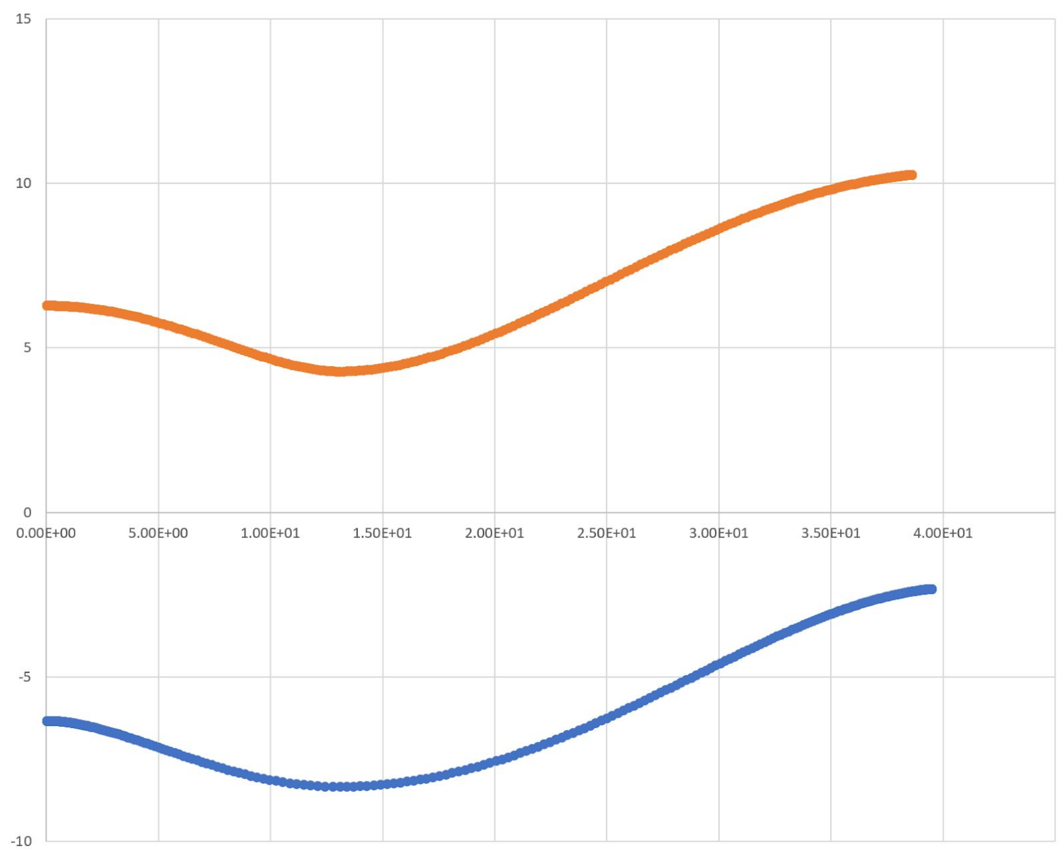


Figure 1-2

Motion Profiling

- To allow the robot to follow a spline or move accurately, we use a Jerk Limited Motion Profile.
- Each trajectory stores position, velocity, acceleration, jerk, heading, time interval between value.
- These values are passed to the drivetrain **encoders** and **gyros** through the use of a Proportional Derivative Velocity Acceleration controller for the **encoders** and a Proportional controller for the **gyro**.

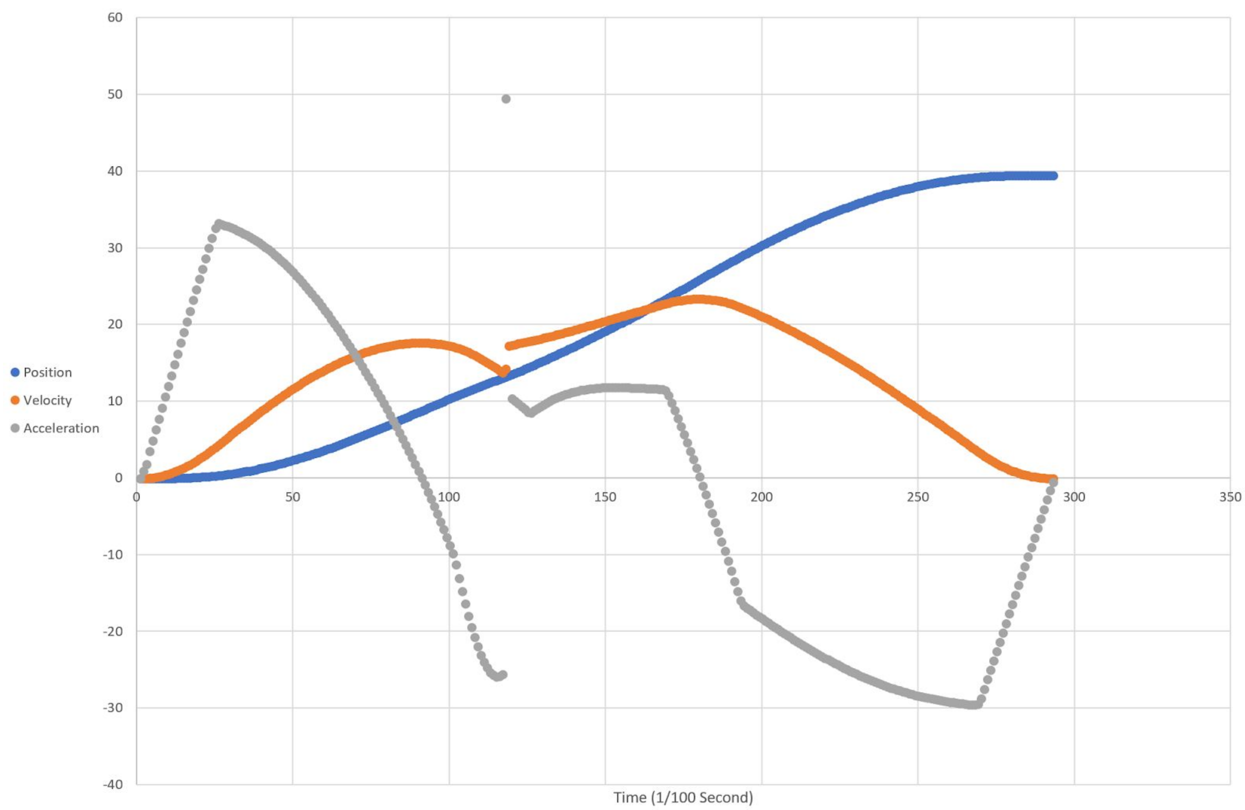


Figure 1-3

- We use feed-forward and feedback control in the form of a Proportional Derivative Velocity Acceleration (PDVA) controller in order to follow this trajectory.
- To calculate the output to the motor we use the following equations:

$$error = segment.Position - currentPosition$$

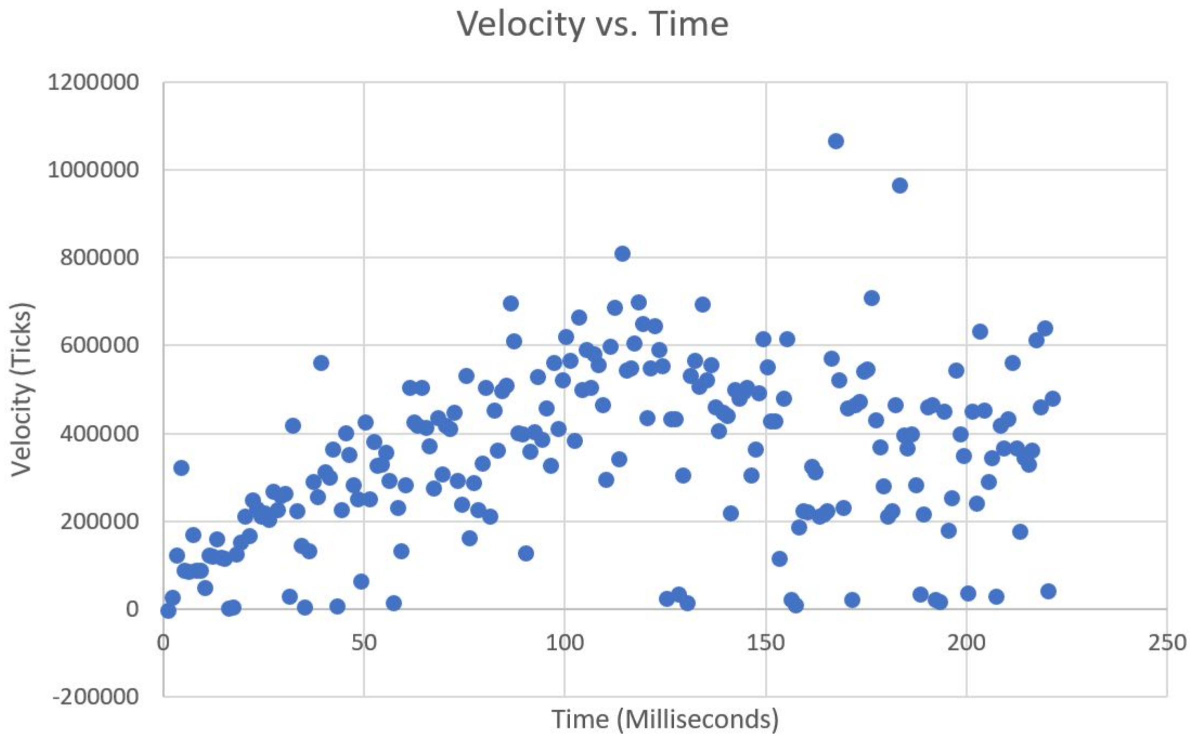
$$Motor\ Output = (kP * error) + (kD * \frac{error - lastError}{segment.DT - segment.Vel}) + (kV * segment.Vel) + (kA * segment.Acc)$$

- The kV and kA do 90% of the control of the motors. kP and kD are used to account for differences in voltage, friction, and uncontrollable factors.
- In order to find kV and kA, we found the max velocity and max acceleration of the robot. See Graph 1-1 for the data that we used to find the max velocity.

$$kV = \frac{1}{max\ Velocity}$$

$$kA = \frac{1}{max\ Acceleration}$$

- This algorithm uses feedforward control to predict the future position of the robot. Sensors are used passively to monitor and fix robot position.
- Use cases highlighted in **green**.



Graph 1-1

- Table 1-1 Shows how this algorithm handles sensor failure when following a motion profile

Failure Case	**Fallback to handle Failure**
If both **gyros** fail	Robot still follows the correct path, within ±0.25in, just using **encoders** using feedback and feed forward control
If both **encoders** fail	Robot still follows the correct path, within ±0.75in because of the feed-forward control
If both **gyros** and both **encoders** fail	Robot follows the path, within ±1.0in, because of feed-forward control

Table 1-1 Motion profiling Fallback

Jewel Detection

- Uses **phone camera** to determine the color of the jewel closest the the VuMark. This is determined at the same time the VuMark is scanned.
- Vuforia converts frames into a bitmap which is scaled down. Then the bitmap is scanned in a tuned, predesignated area for RGB values. This determines whether the glyph is red or blue.
- Faster than the OpenCV vision library because it uses the native Vuforia library included in the FTC SDK.
- More accurate than a color sensor since it can scan a much wider area for a longer period of time and is not affected as much by varying lighting conditions.
- We have found this to be much more effective in competitive events (9/9 jewels during East Super-Regional qualifying rounds compared to 3/5 jewels during New Jersey Central League Championship).
- Figures 1-5 shows an example of the bitmaps that were captured and saved. Note that the bitmaps are scaled down to 36 by 64 pixels. This lowers file size without compromising accuracy.
- Figure 1-6 shows the process of how we scan for the jewels during initialization.

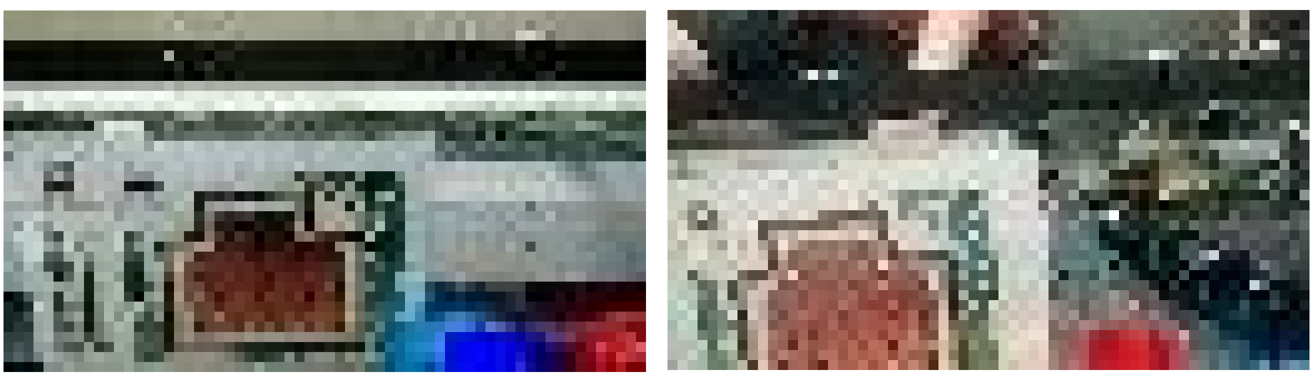


Figure 1-5

Failure Case	**Fallback to handle Failure**
Scans may be inaccurate	Multiple Vuforia frames are converted to bitmaps and scanned
Jewel color cannot be determined	Jewel knocking routine is skipped

Table 1-2 Fallback for Jewel Detection

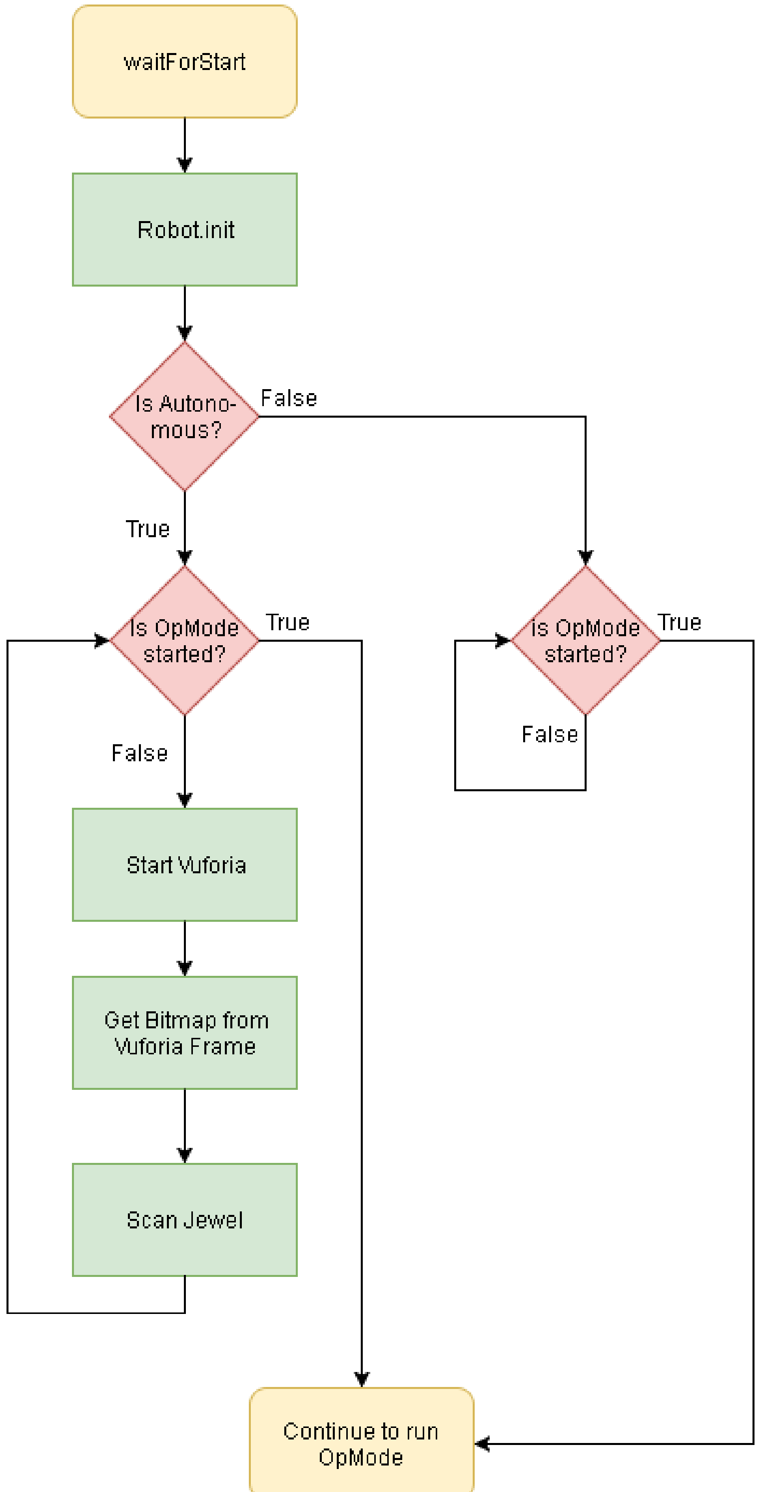

waitForStart
Robot.init
Is Autono-mous?
False
True
Is OpMode started?
True
False
Start Vuforia
Get Bitmap from Vuforia Frame
Scan Jewel
Is OpMode started?
True
False
Continue to run OpMode

Figure 1-6

Glyph Tray Detection

- Uses both **optical distance sensors**
- **Optical distance sensors** are placed to find glyphs in different positions of the tray. They are also used to find the color of glyphs.
- Figure 1-7 shows the position of the **optical distance sensors**

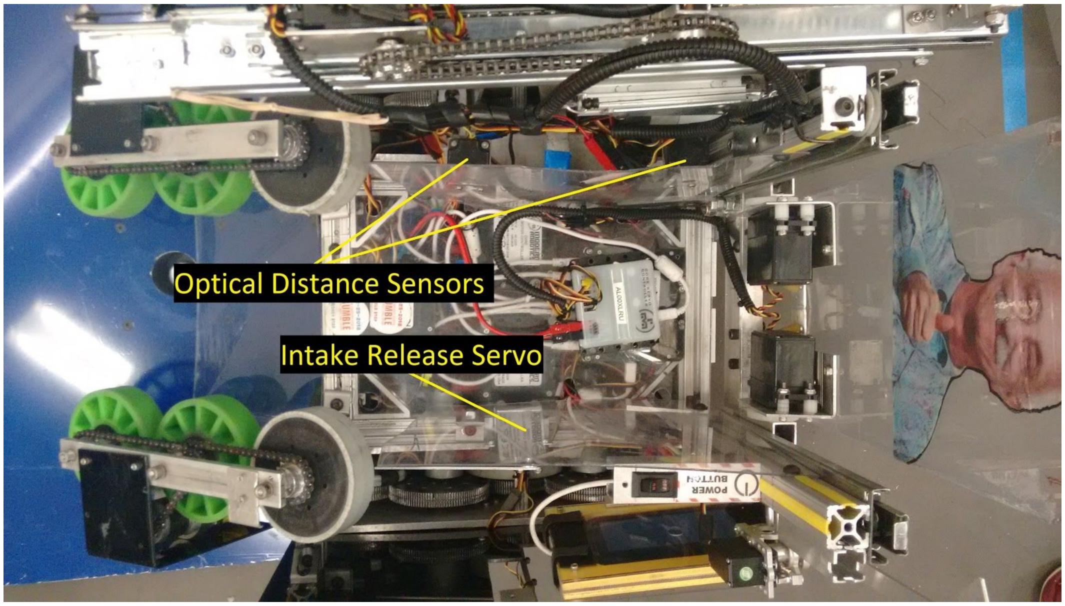


Figure 1-7

Failure Case	Fallback to handle Failure
Optical Distance Sensors are not working	The intake will run in a strategic pattern with the glyph stopper down before returning to the cryptobox to maximize the chance of intaking glyphs without sensors.

Table 1-3 Fallback for Glyph Tray Detection

Autonomous Path/Flow

1. Scan VuMark and jewel color
    a. Both the VuMark and the jewel color are scanned at the same time with Vuforia through the **phone camera**
        i. The VuMark is determined through the Vuforia library.
        ii. The jewel color is scanned
        iii. See Jewel Detection
2. Score the correct jewel based on the color of the jewel scanned
    a. If the jewel that needs to be knocked off is the one closest to the cryptobox, the jewel arm is left down when driving off the stone to save time.
3. Drive to the cryptobox based on VuMark
    a. Uses the **path generation** and **motion profiling** algorithm described above
    b. See Path Generation and Motion Profiling
4. Score the cryptokey in the correct column
5. Retrieve extra glyphs
    a. Follow a **pre-generated Cubic Hermite spline** to the glyph pit
    b. Based on the data values of the **optical distance sensor**, the robot will either stay in the glyph pit to intake more glyphs or return to the cryptobox
    c. See Glyph Tray Detection
6. Score extra glyphs
    a. Follow a **pre-generated Cubic Hermite spline** and **motion profiling** back to the cryptobox
    b. If there is extra time, the program will repeat steps 5 and 6.
    c. If not, it will park in the safe zone.

Figure 1-8 shows the autonomous path for each position.

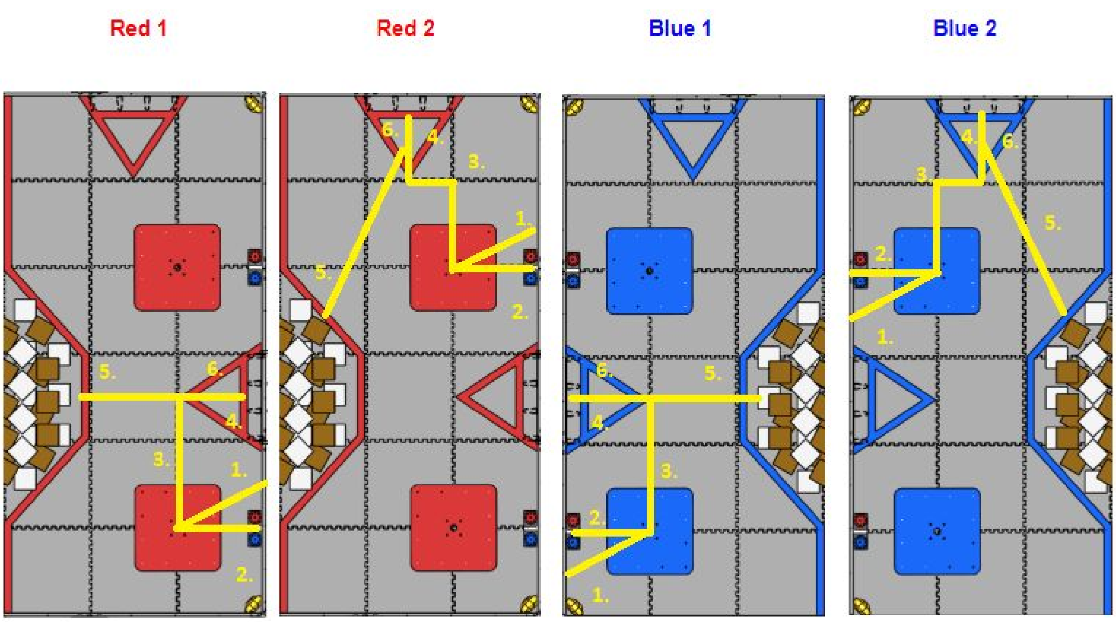


Figure 1-8