Trajectory Following

Course Overview

Course Overview

- Different methods to get from here to there
- Geometry – Describe position of one thing relation to another thing.
- Kinematics – Define robot physical characteristics used to convert between robot movement and wheel movement.
- Odometry – Track where the robot is.
- Trajectory Creation – What are and how to create trajectories.
- Trajectory Execution – How to execute a trajectory

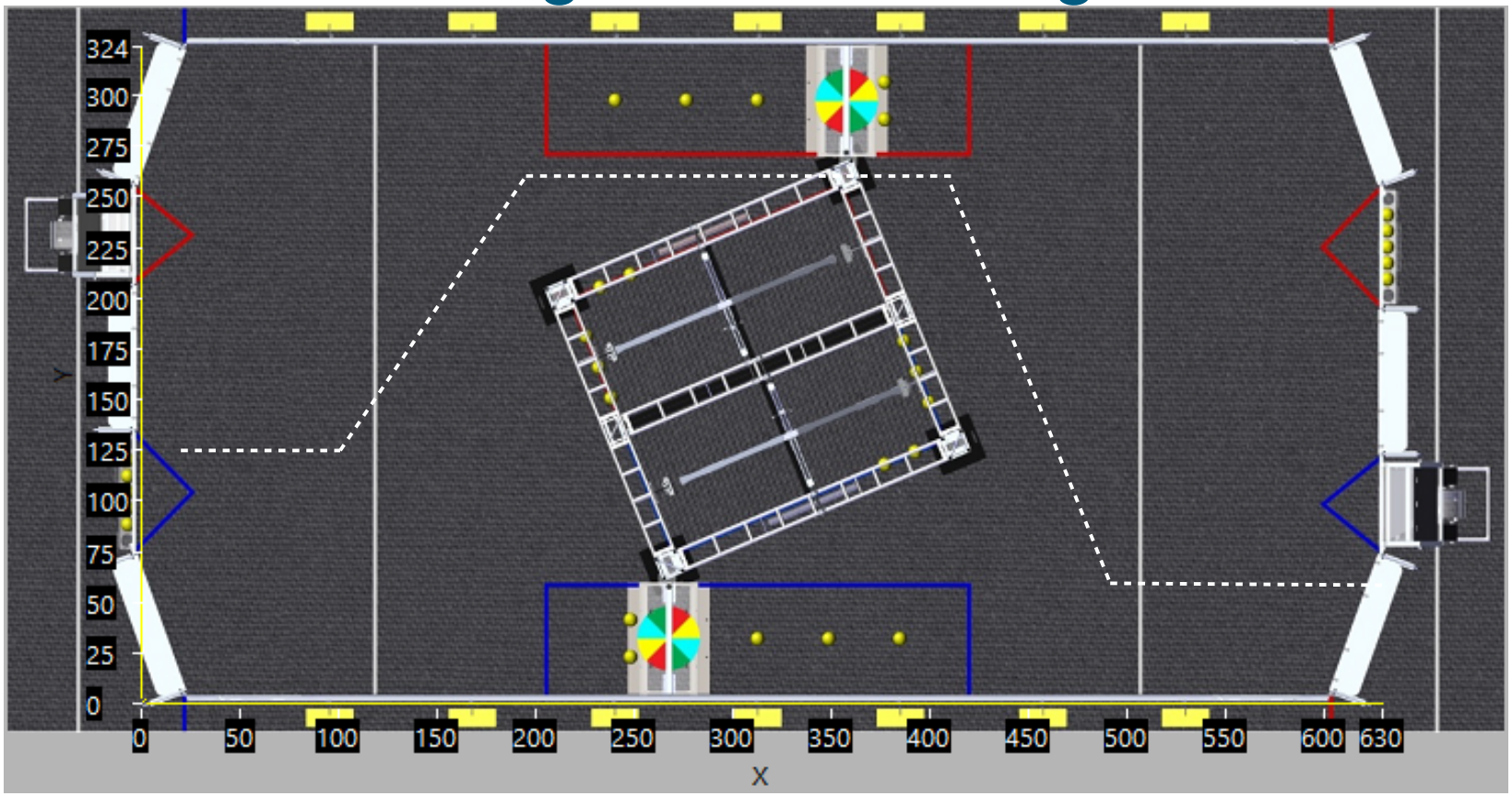
Trajectory Following

Getting from Here to There

Getting from Here to There

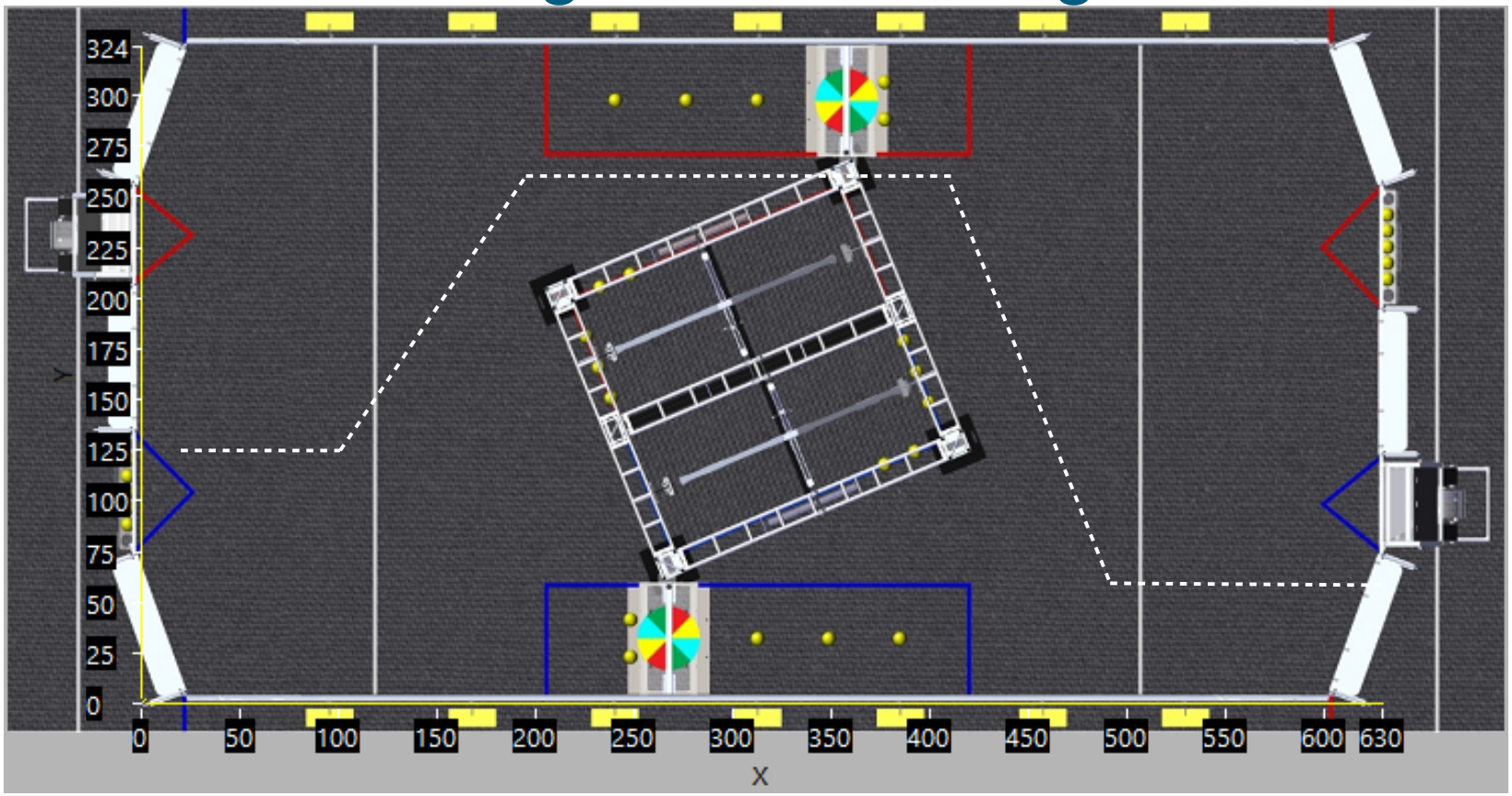
- **Different methods to get from one place to another without driver intervention.**
  - Could be used during autonomous or teleop competition phase

Method 1 - Straight, Turn, Straight



- Series of straight and turn (spin) movements.
- Each movement, either go straight, or spin, starts at zero velocity and ends at zero velocity.
- Error of each movement can be minimized. These errors accumulate.

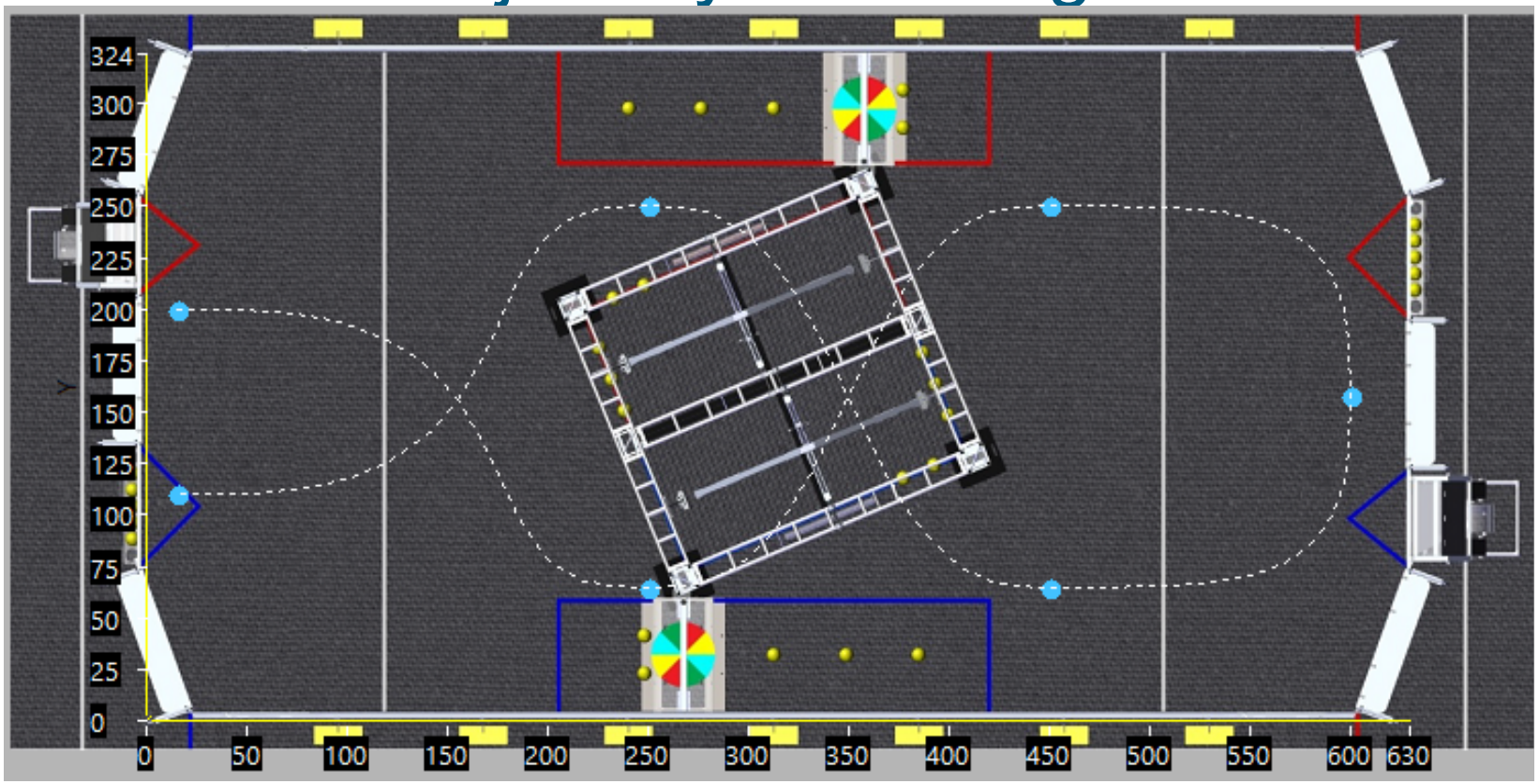
Method 1 - Straight, Turn, Straight



Advantages - Software easy to implement. Easy to instrument. Easy to change. Easy to tune. Can be tuned to be accurate.

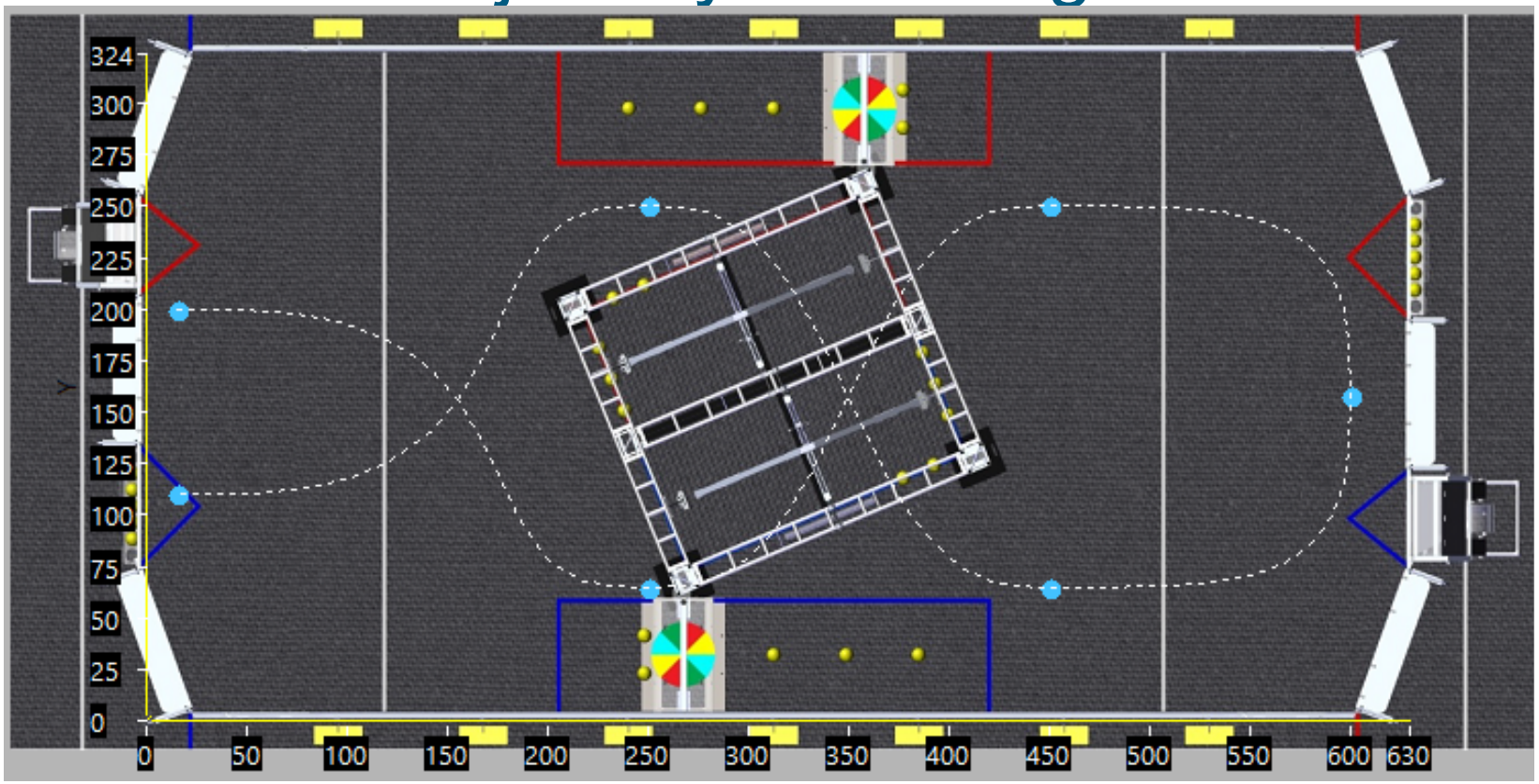
Disadvantages – Potentially slower execution – Each move has to come to a stop before making the next move.

Method 2 - Trajectory Following



- A single continuous movement.
- Velocity is constrained as needed by physical robot characteristics.
- Error of sensors accumulates over time.

Method 2 - Trajectory Following



Advantages - Potentially faster execution (no stopping along the way). Can follow more complex paths. Can be accurate.

Disadvantages – Software harder to implement. More instrument coordination. Harder to change. Harder to tune.

Method Comparison

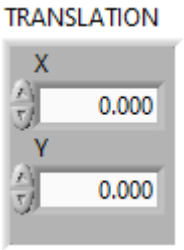
	Straight / Turn	Trajectory
Sensors	Distance (encoders), Heading (gyro) Gyros drift over time.	Same plus Vision Measurements can also be used. (Pose Estimation)
Accuracy	Position error of each movement accumulates	Position error increases with time of trajectory and inaccuracy of speed control.
Implementation	Doesn't require good speed control to be accurate.	Requires good speed control to maintain accuracy.
Tuning	Once straight and turn routines are tuned, combinations shouldn't need to be tuned.	Robot physical characteristics – effective track width, max speed and acceleration need to be measured or modeled.
Competition Changes	Easier – Changes don't need to be re-tuned	Possibly harder – Testing and tuning might be needed.
Execution Speed	Execution is slowed by each movement coming to a stop. Speed is also a function of tuning.	Execution speed is improved by a single continuous movement. Speed is also a function of tuning.

Trajectory Following

Geometry

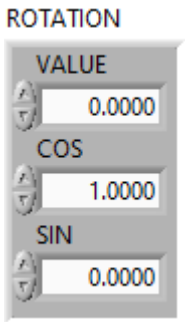
Location - Translation2D

- **Specified by X, and Y**
  - In relation to robot, X increases forward, Y increases to the left.
  - In relation to a field drawing X increases to the right, Y increases up.
  - Units are SI (meters). Any units can be used, but when calling the library routines, they expect SI units. (Different languages deal with units somewhat differently.)
- **Translation2D Data**



Heading / Orientation – Rotation2D

- **Which way the front of the robot is facing. This is also the direction of travel.**
  - Specified by angle of rotation
  - Values increase – counterclockwise. The normal gyro reading in previous years has been backwards from this!
  - Units are SI (radians). >> Sometimes degrees – read the documentation!
- **Rotation2D Data**
  - Value is angle. Sin and Cos are stored so they don't have to be recalculated.

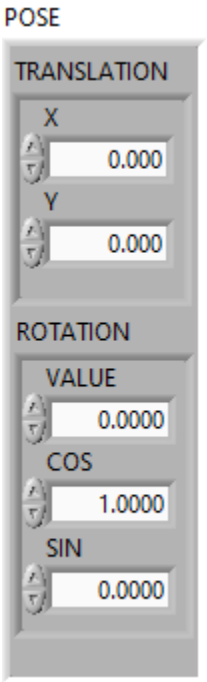


Heading / Orientation – Robot vs Travel

- **Robot orientation and direction of travel**
  - For robots with Swerve or Mecanum drives the orientation of the robot does not have to match the direction of travel.

Position – Pose2D

- **Pose2D - Contains both Translation2d and Rotation2d**



Absolute and Relative Position

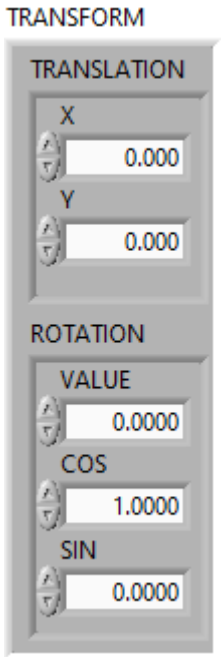
- **Pose2d can be specified as absolute or relative.**
  - Absolute Location most often refers to a fixed point on the field, usually one of the corners as (0,0). This can vary, but is usually the bottom left of the field (pathfinder and the trajectory library use this, Pathweaver uses the top left.) Whatever is used, be consistent.
  - Absolute Orientation and Heading angle, origin (0), always both the same, point in the X direction.
  - Relative Location means the origin is relative to something else. Often the origin is the current location of your robot. (For Mecanum and Swerve robots the physical center of the robot is used.)
  - For robot relative, the orientation is 0 in the direction that the front of the robot faces.

Geometry Math

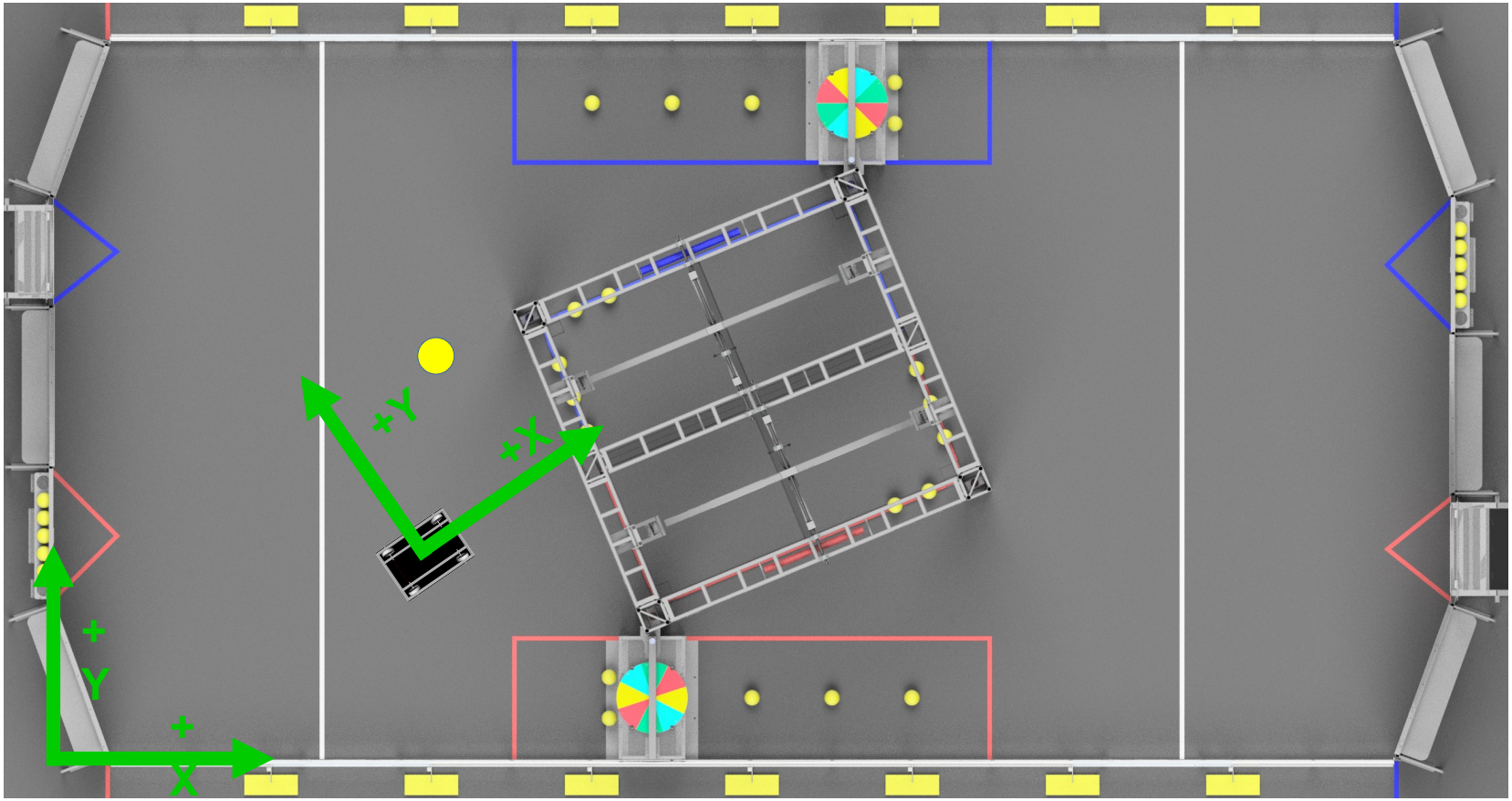
- **Translation2D**
  - Add, Subtract, Scalar multiply, Scalar divide, Rotate
- **Rotation2D**
  - Add, Subtract, Scalar multiply, Scalar divide
- **Transformation2D**
  - Operates on a Pose2D
  - Contains a Translation2D and Rotation2D
- **Twist2D**
  - Represents the change in distance along an arc.
  - Used internally in Odometry

Transformation 2D

- **Transformation2D**
  - Acts upon a Pose2D
  - Transforming a Pose2d by a Transform2d rotates the translation component of the transform by the rotation of the pose, and then adds the rotated translation component and the rotation component to the pose.
- **Transform2D Data**



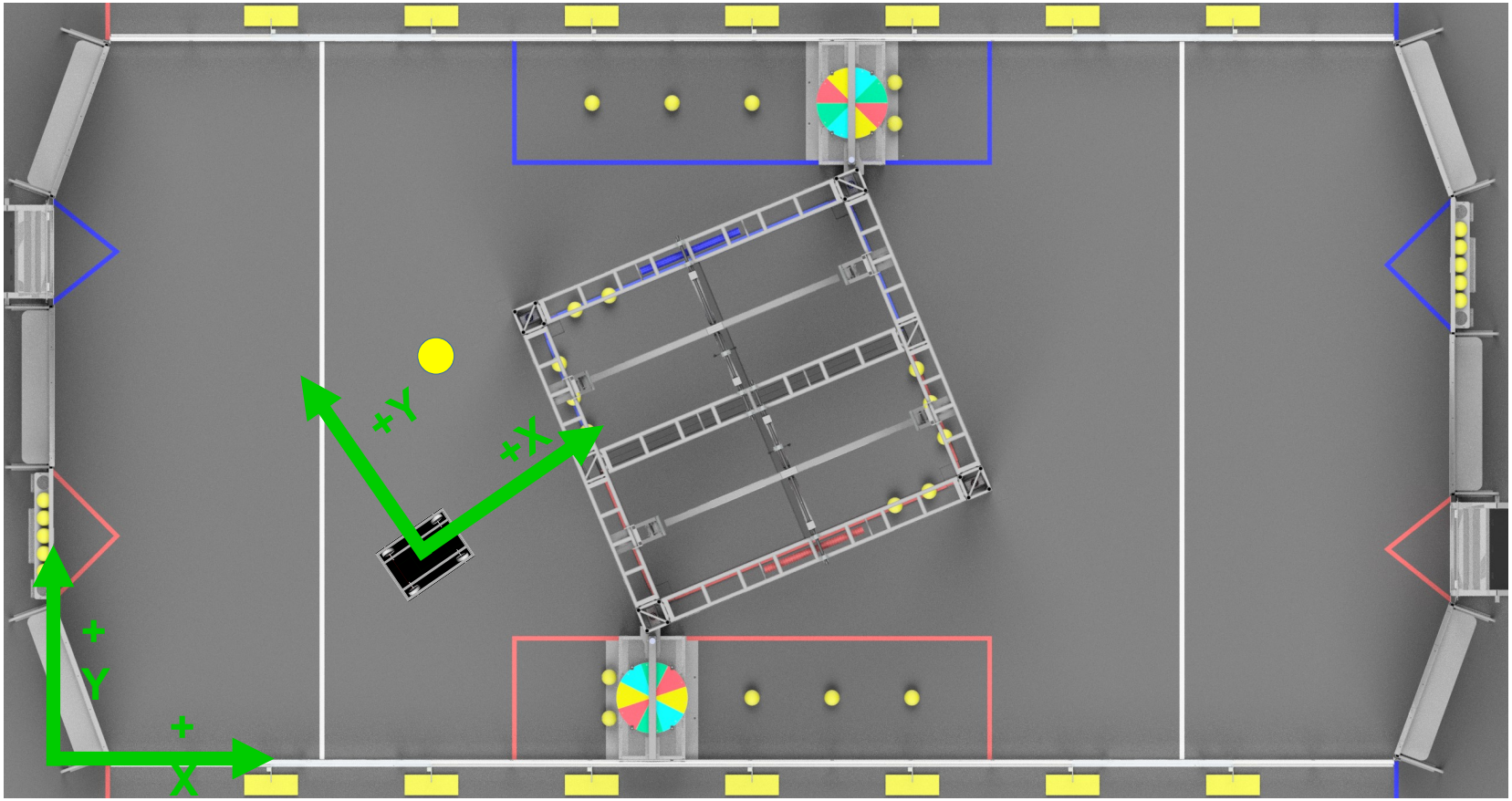
Absolute Geometry Example



- Absolute robot position (approx.) 15 ft, 10 ft, 40 deg. (X, Y, heading) [Pose2D]
- Absolute ball position (approx.) 15 ft, 18 ft. (X, Y) [Translation2D] (Ball has no rotation.)

Note: Units here are feet and degrees. All the trajectory Trajectory Lib routines, use meters and radians. Conversion must be performed prior to calling them.

Relative Geometry Example



- Robot relative robot position 0 ft, 0 ft, 0 deg. (X, Y, heading) [Pose2D]
- Robot relative ball position (aprox.) 3 ft, 4 ft. (X, Y) [Translation2D] (Ball has no rotation.)

Note: Units here are feet and degrees. All the trajectory Trajectory Lib routines, use meters and radians. Conversion must be performed prior to calling them.

Demo

- **Show geometry_demo.**

Geometry – Additional Information

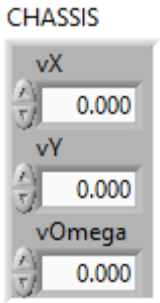
- https://docs.wpilib.org/en/stable/docs/software/advanced-controls/geometry/index.html

Trajectory Following

Kinematics

Kinematics – Translating Robot to Wheels

- **Kinematics – Translating between robot movement and wheel movement.**
- **Robot Movement uses Chassis Speed**
  - Consists of X, Y, and rotational movement.
  - vX – Forward, backward movement (m/s)
  - vY – Side side movement (m/s)
  - vOmega (rotational) – How much the robot is turning. (radians/sec). Counterclockwise is plus.
    - Differential drive robots always have have a Y movement of 0.0.



Kinematics Data

- **Differential Drive**
  - Track Width – This is the effective width between wheels. Generally this is 1.0 to 2.5 times the actual track width.
    - Find this by testing. Spin robot X degrees. Measure wheel distance traveled. This is the circumference. Calculate diameter. This is effective track width.
- **Mecanum, Swerve Drive**
  - Location of each wheel relative to center of robot. These are actual measurements.
- **Units are all SI (meters)**
- **Values increase – forward, left, counterclockwise**

Conversion to from wheel speeds

- **Convert desired ChassisSpeed to Wheel speeds**
  - Routines exist to do this for each different type of standard drive train – differential, swerve, mecanum.
- **Convert wheel speeds to ChassisSpeed**
  - Routines exist to do this for each different type of standard drive train – differential, swerve, mechanum.

Kinematics – Additional Information

- https://docs.wpilib.org/en/stable/docs/software/kinematics-and-odometry/index.html

Trajectory Following

Odometry

Odometry – Measuring Robot's Position

- **Odometry – The process of measuring the robot's position.**
  - Absolute or relative field position.
    - An absolute position needs an accurate starting position and heading [Pose2D].
  - Calculates updated position from robot sensors – wheel distance (encoders), and heading (gyro)
  - Robot position accuracy gets worse over time, even if the sensor measurements are calibrated really well. (Even little error accumulate.) As such Odometry can generally only be used for short periods of time. This is generally enough to accurately execute a trajectory.

Odometry - Process

- **Set initial position**
  - Set to a known absolute position or set to 0,0,0 for relative movement.
- **Periodically call the Update routine**
  - Provide Gyro and drive encoder readings. Update routine will calculate new robot position.
- **Notes:**
  - Each robot drive type (differential, mecanum, swerve) has their own routines.
  - Some documentation recommends resetting gyro and encoder readings. Be careful of this!!! If these readings are used elsewhere, those routines will likely break.
  - Multiple instances of odometry may be running at the same time. For example one for position since start of match and one for position since start of trajectory execution.

Odometry - Issues

- **Gyro Sensor Drift**
  - Gyro readings drift over time. Can only use for a limited time. Good gyro calibration required.
- **Drive wheel encoder calibration**
  - Encoders need to be carefully calibrated so each is accurate.
- **Robot “skid”**
  - Robot wheels skid while turning and potentially while accelerating and decelerating.
- **Initial robot placement**
  - Initial position and rotation of robot is very important. Rotational errors get worse as distance increases!

Odometry – Additional Information

- https://docs.wpilib.org/en/stable/docs/software/kinematics-and-odometry/index.html

Trajectory Following

Trajectory Basics and Creation

Trajectory – Basic Data Structure

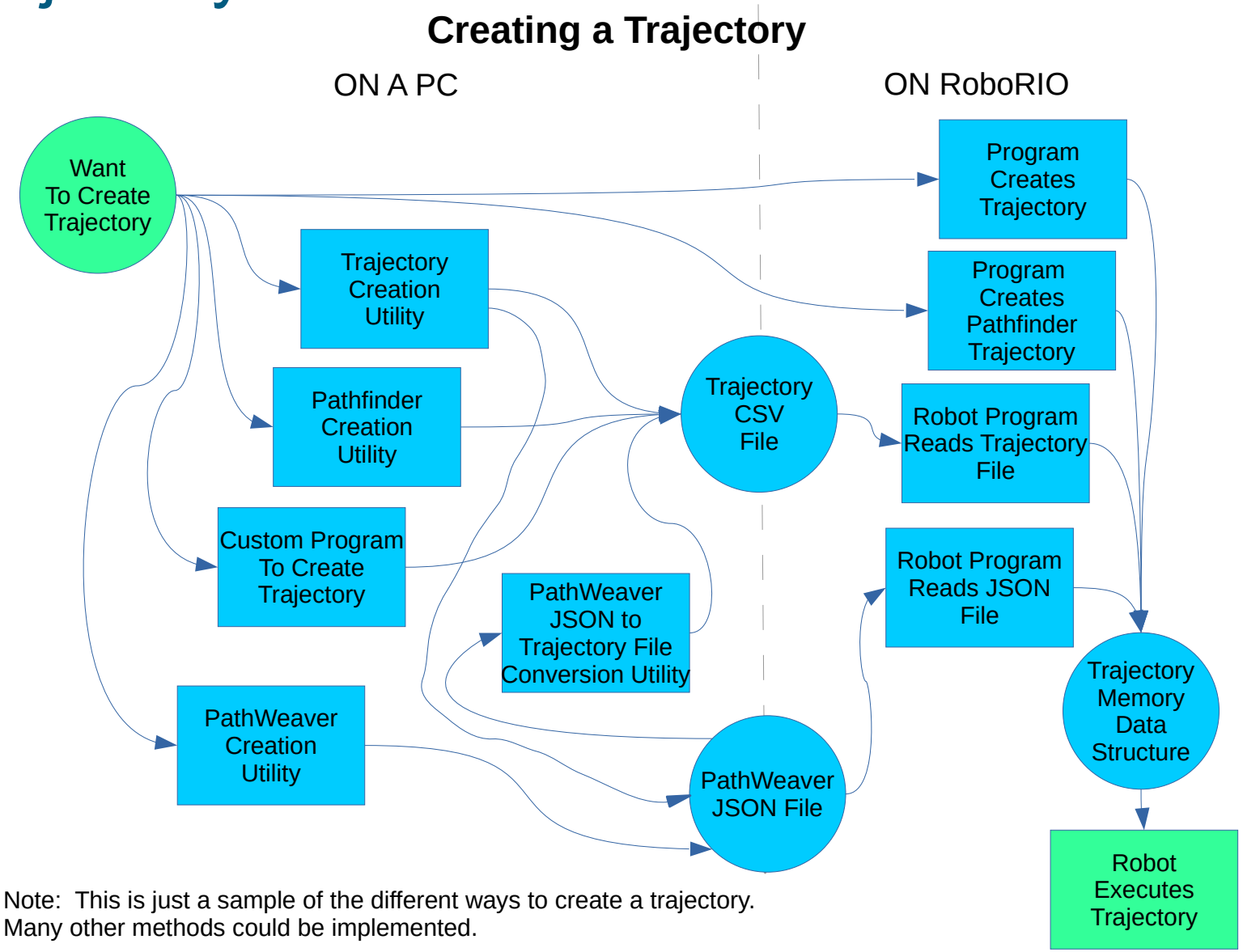
- **A Trajectory is a list of relative times, expected positions, and desired robot chassis speed**
- **Read into memory. Robot samples by time.**
- **Time interval isn't constant. Library routines use interpolation to sample a specific time.**

2021-sample-challenge-quintic.csv - Notepad

File Edit Format View Help

```
#----------------------------------------------------------------------------------------
#-- Time,       Velocity,   Accel,      X pos,      Y pos,      Heading, Curvature,
#-- Seconds,    m/s,        m/s^2,      m,          m,          radians,    radians/meter,
   0.00000,    0.00000,    0.63500,    0.00000,    0.00000,    0.00000,    0.00000,
   0.45081,    0.28626,    0.63500,    0.06453,   -0.00002,   -0.00075,   -0.02304,
   0.63738,    0.40474,    0.63500,    0.12899,   -0.00013,   -0.00295,   -0.04519,
   0.78032,    0.49550,    0.63500,    0.19332,   -0.00043,   -0.00655,   -0.06664,
   0.90053,    0.57184,    0.63500,    0.25748,   -0.00100,   -0.01150,   -0.08754,
   1.00613,    0.63889,    0.63500,    0.32139,   -0.00193,   -0.01775,   -0.10804,
   1.10124,    0.69929,    0.63500,    0.38502,   -0.00329,   -0.02527,   -0.12825,
   1.26902,    0.80582,    0.63500,    0.51120,   -0.00761,   -0.04398,   -0.16814,
   1.41528,    0.89870,    0.63500,    0.63567,   -0.01449,   -0.06741,   -0.20773,
   1.54593,    0.98167,    0.63500,    0.75810,   -0.02443,   -0.09535,   -0.24724,
   1.66450,    1.05696,    0.63500,    0.87822,   -0.03783,   -0.12760,   -0.28658,
   1.77336,    1.12608,    0.31827,    0.99579,   -0.05504,   -0.16396,   -0.32533,
```

Trajectory Creation / Use - Flowchart



Note: This is just a sample of the different ways to create a trajectory. Many other methods could be implemented.

Trajectory – Defined

- **Created from “waypoints”**
  - A list of positions along the path. Must include the starting and ending positions. Can include other positions along the path.
  - Optionally weights are defined for each point. This determines how “straight” the path goes through a particular point.
- **Uses Robot Kinematics**
- **Constraints define robot physical limitations**
  - Max speed, max acceleration, max centripetal acceleration, etc.
  - Can be estimated or obtained through robot characterization testing.
  - Start slow and increase speed and acceleration until accuracy and repeatability become intolerable.

Trajectory Creation Process

- **Library routines do all this work. This is the process.**
  - Create smooth path between waypoints using splines
    - There are many curvefitting techniques. Splines are simple and fast. This does NOT include timing or robot speed information.
  - Create trajectory by calculating desired robot speed along path ensuring constraints are not violated.
    - Sample times, velocities, acceleration, and curvature are added.
    - This is a multi-pass process.
  - Populate memory data structure and/or write to a file.

Trajectory Optimization

- **Test Trajectory Execution and Revise Constraints.**
  - Constraints and robot parameters may need to be adjusted until trajectory execution is optimized.
- **Hints**
  - May need to artificially change effective track width dimension to accommodate “skid” while turning.
  - Begin with slow maximum speed and revise to faster speeds.
  - Disable closed loop, ramsete, control to find best trajectory constraints.
  - Do testing to find maximum speed, acceleration, and turning speed.
    - There is a characterization program. Read about it. The data will have to be translated to whatever tool is used to create trajectories.

Trajectory Creation Demo

- **Demo trajectory creation utility.**
- **Other utilities exist:**
  - PathWeaver
  - PathFinder
  - Some teams made their own.

Trajectory Creation – Additional Information

- Documentation

https://docs.wpilib.org/en/stable/docs/software/wpilib-tools/pathweaver/introduction.html

- Trajectory Creation Tool Download

https://github.com/jsimpso81/FRC_LV_TrajLib_Utilities

Trajectory Following

Trajectory Execution

Controlling Trajectory Execution

- **Different controllers for different drive types**
  - Differential Drive – Ramsete Controller. This uses a “uni-cycle” controller. The default tuning should be sufficient.
  - Mecanum, Swerve – Holonomic Controller. This combines X, Y, and rotation PIDs to set drive setpoints and correct for trajectory error. The PIDs will need to be tuned.
- **Used as a front end to create drive setpoints, not as a replacement of speed control.**

Controlling Trajectory Execution - Ramsete

- **Inputs**
  - Trajectory sample for the time offset.
  - Current Odometry
- **Outputs**
  - Desired chassis speeds. These need to be “normalized” to ensure maximum allowed speed isn’t exceeded and then converted to individual wheel speed demands.
- **Starting Trajectory Execution**
  - Something initiates the execution of a trajectory. This could be an autonomous action or a joystick button pushed by a driver. When initiated, the Odometry data is reset and the relative trajectory time is set to zero.

Controlling Trajectory Execution - Procedure

- **Each execution cycle until the trajectory time has elapsed, the robot control system:**
  - Gets the current elapsed time of the trajectory
  - Updates the odometry to calculate a current robot position.
  - Obtains the trajectory sample for the current elapsed trajectory time. This contains the desired robot position, desired linear and rotation robot velocities.
  - For differential robot's this data is fed to a ramsete control routine which calculates the desired robot Chassis Speeds. Wheel speeds are calculated from Chassis Speeds. These are normalized (to prevent any speed from being larger than the maximum allowed speed.)
  - The normalized Wheel Speed demands are sent to the drive logic, which controls drive wheel speed. This should be done the same as wheel speed is controlled when not executing a trajectory.
- **Notes**
  - Make certain units are converted as needed.
  - Write Odometry, Trajectory sample, Trajectory error, and speed demands to network tables for diagnosis and tracking.

Final demo and questions

Additional Information

- Kinematics / Odometry:

https://docs.wpilib.org/en/latest/docs/software/kinematics-and-odometry/index.html

- Trajectory Tutorial:

https://docs.wpilib.org/en/latest/docs/software/examples-tutorials/trajectory-tutorial/index.html

- LabVIEW Trajectory Library and Samples:

https://github.com/jsimpso81/FRC_LV_TrajLib
https://github.com/jsimpso81/FRC_LV_TrajLib_Util_and_Samp

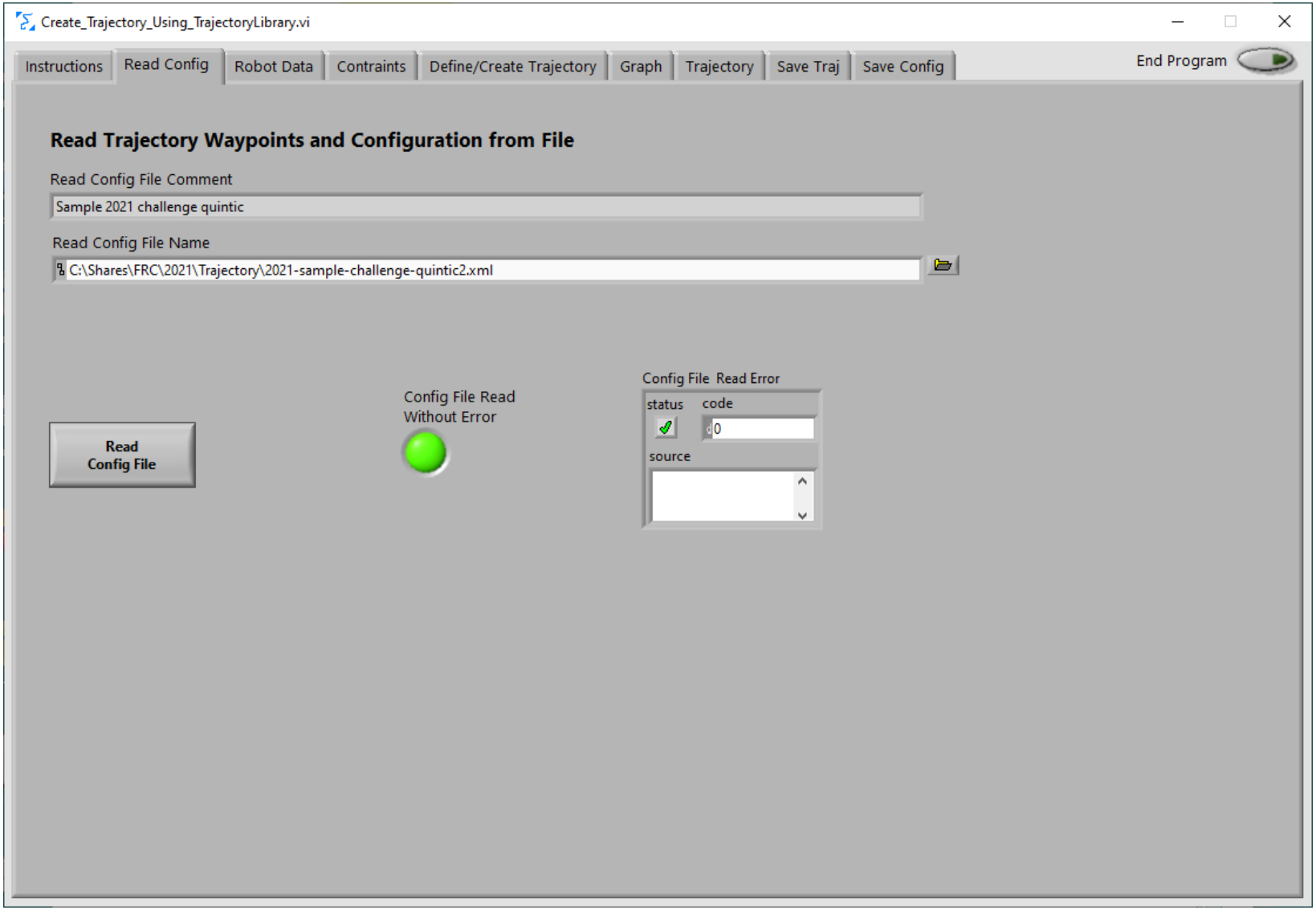
- Team 254 Motion Planning Presentation

https://www.youtube.com/watch?v=8319J1BEHwM

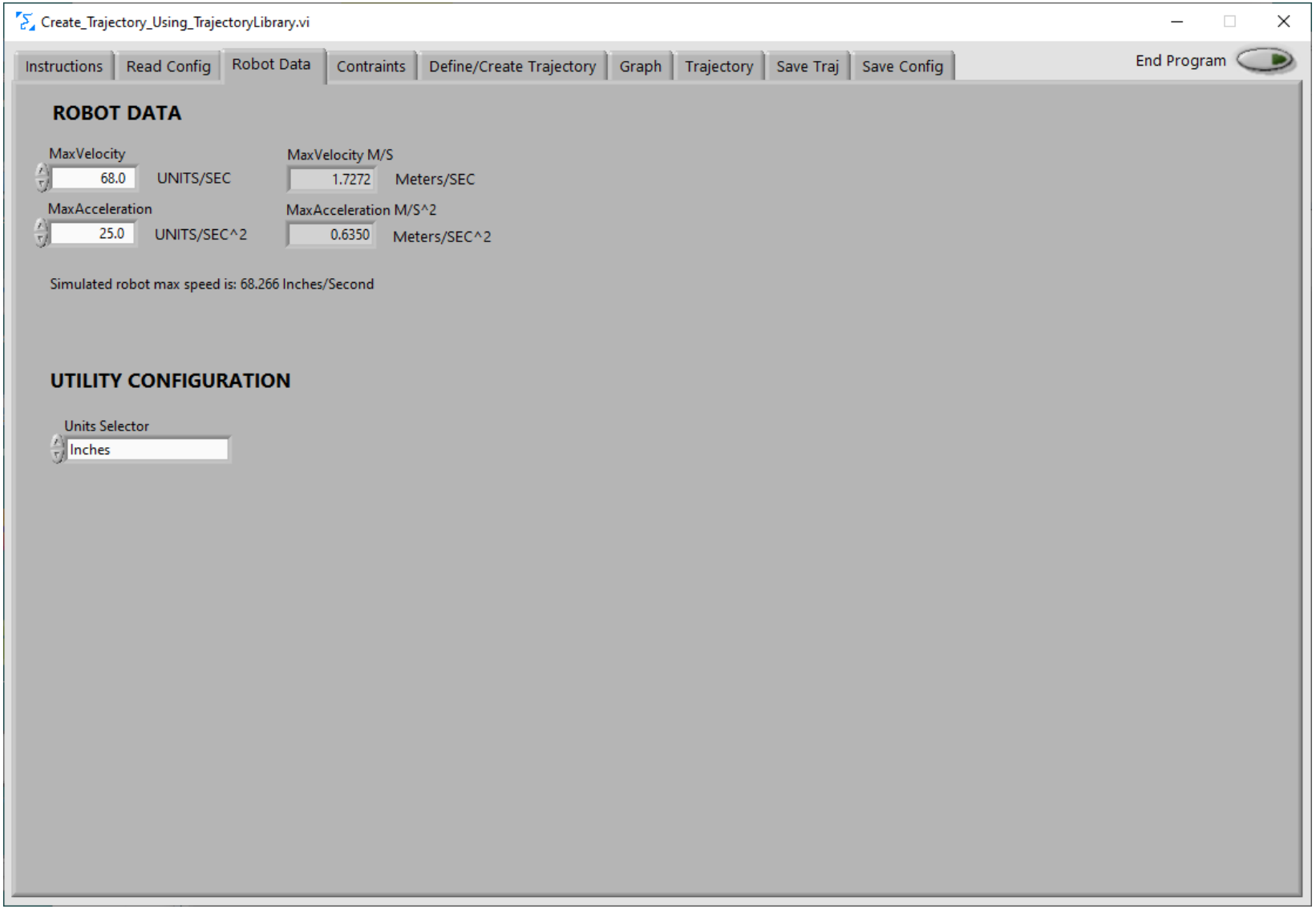
Trajectory Following

Trajectory Creation Sample

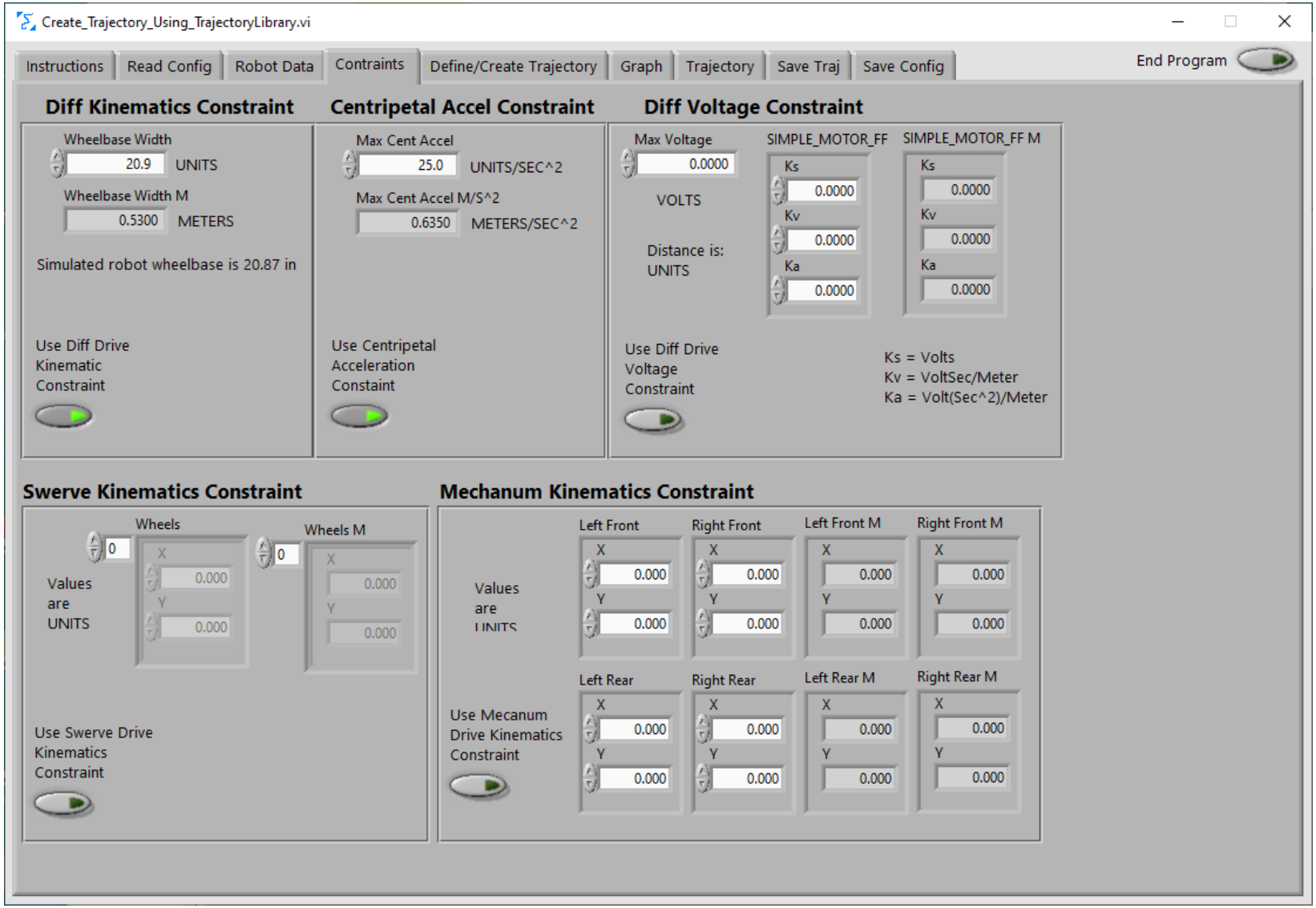
Trajectory Creation – Read configuration



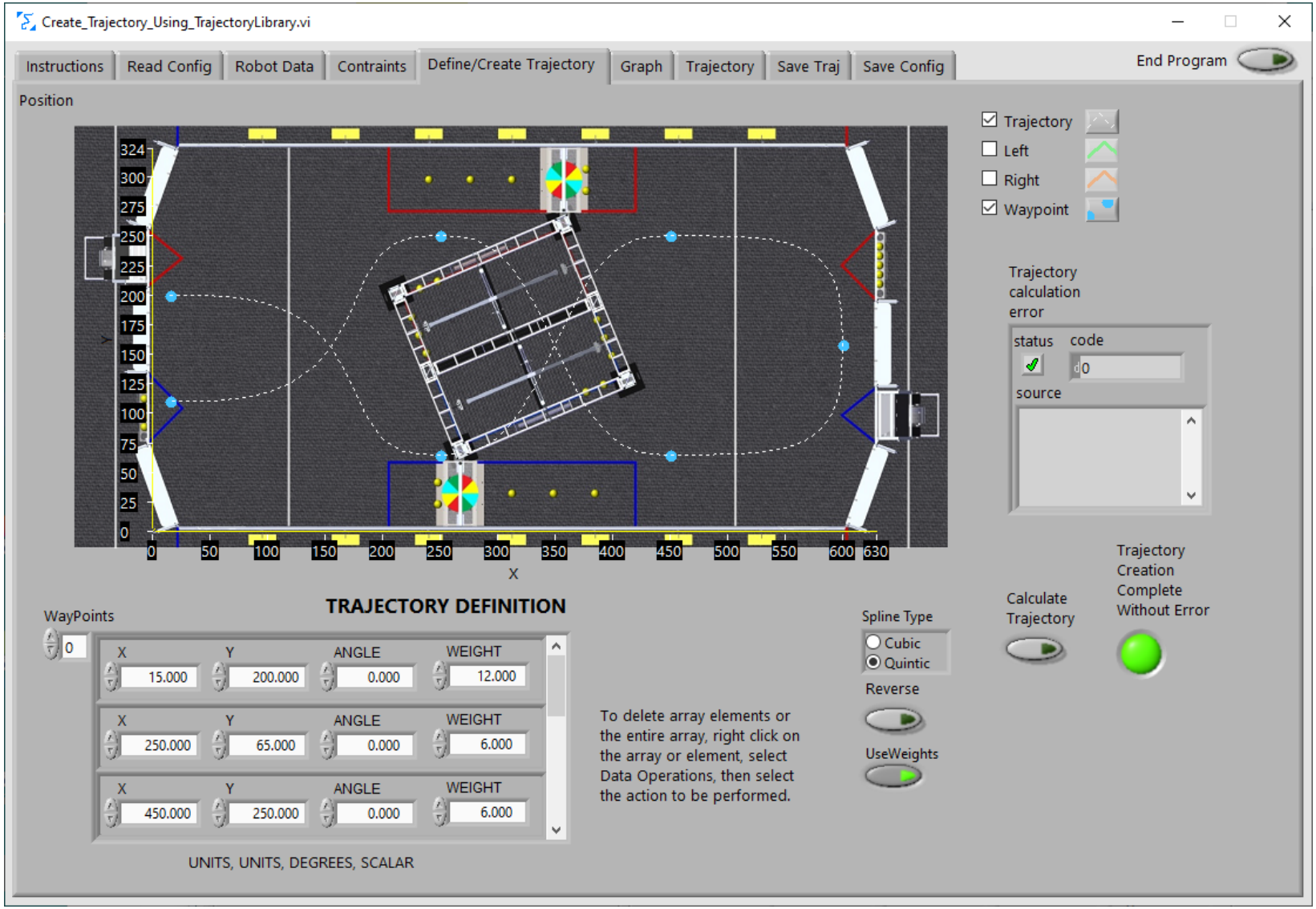
Trajectory Creation – Robot Data



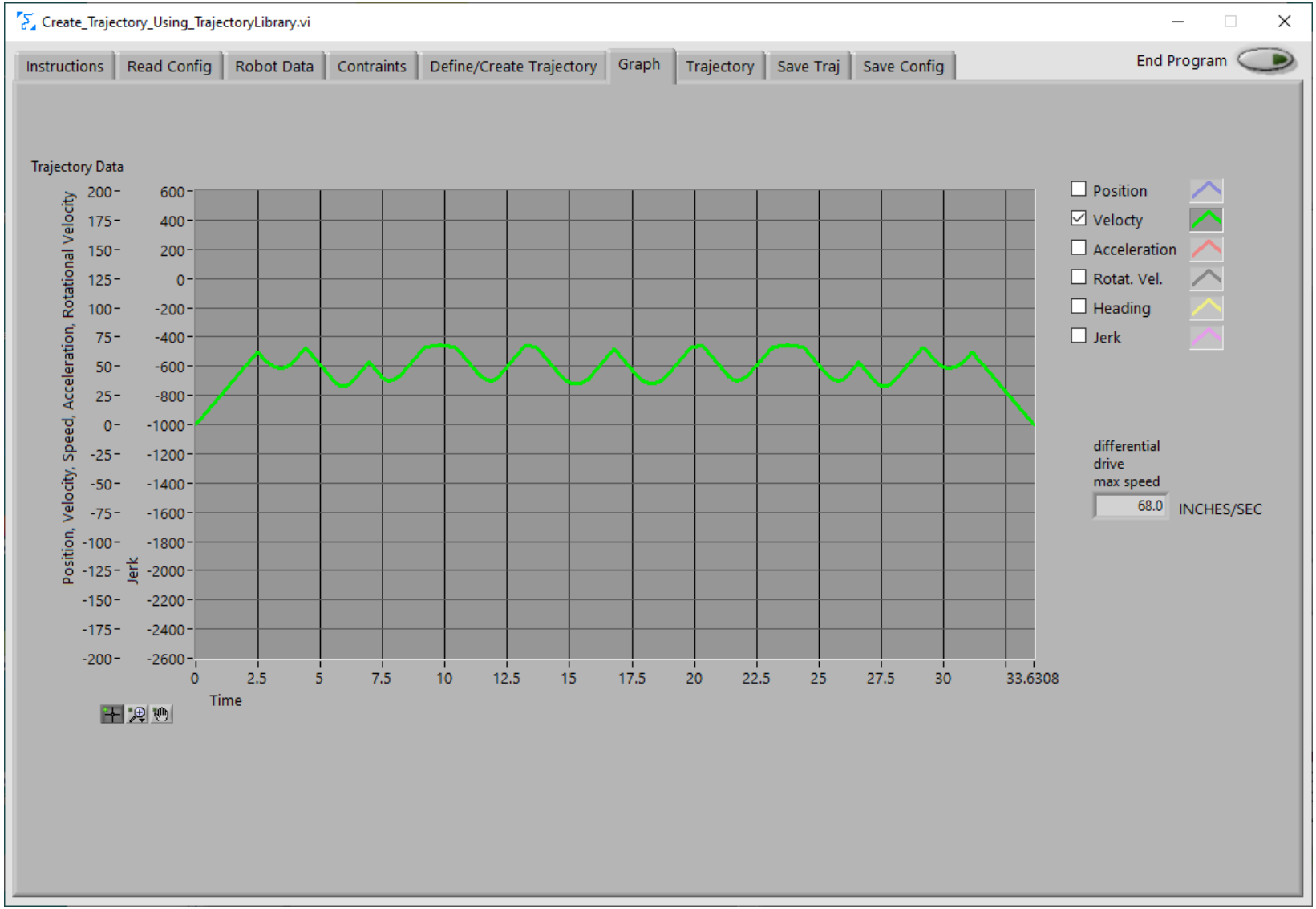
Trajectory Creation – Define Constraints



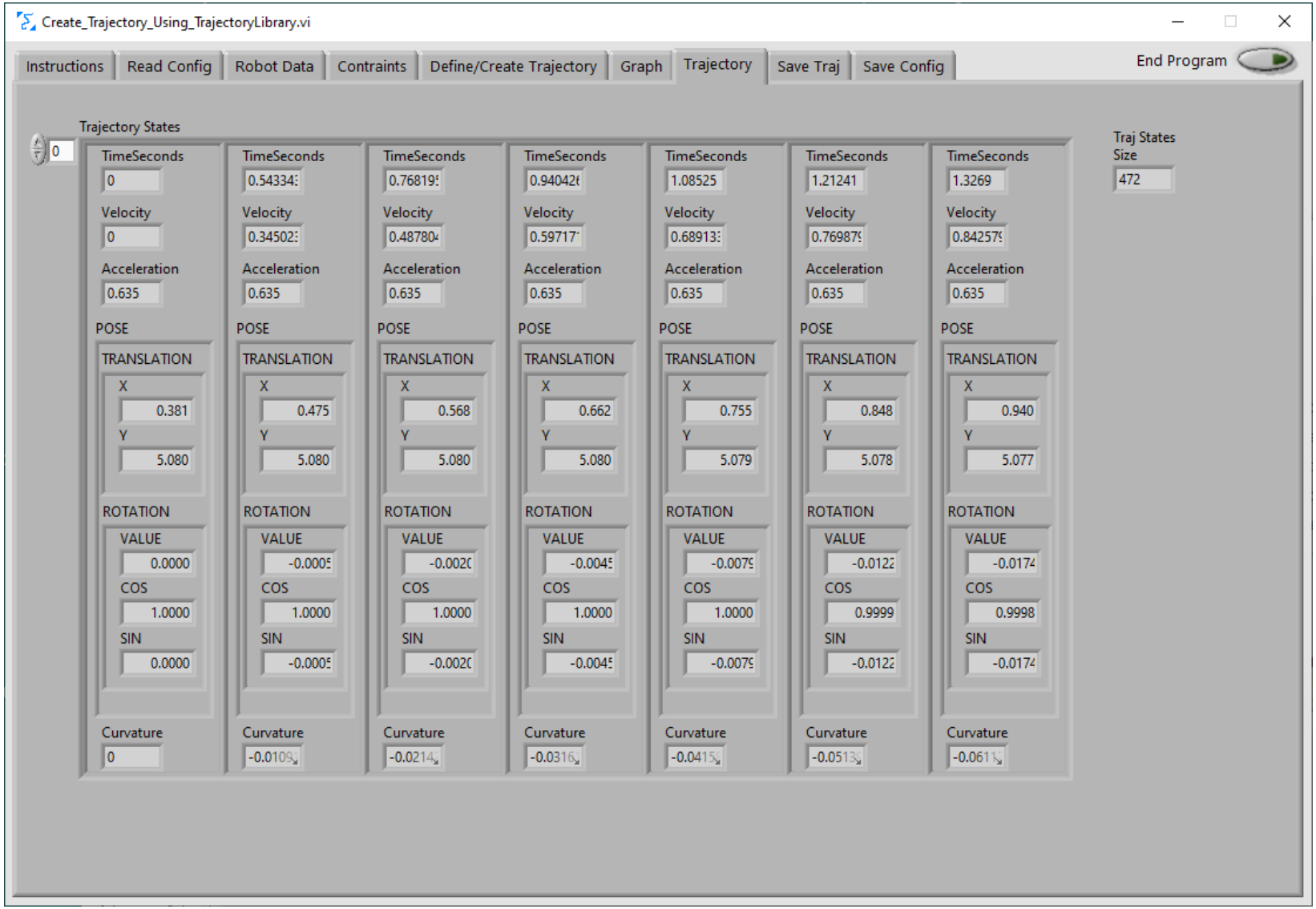
Trajectory Creation – Set Waypoints, Create



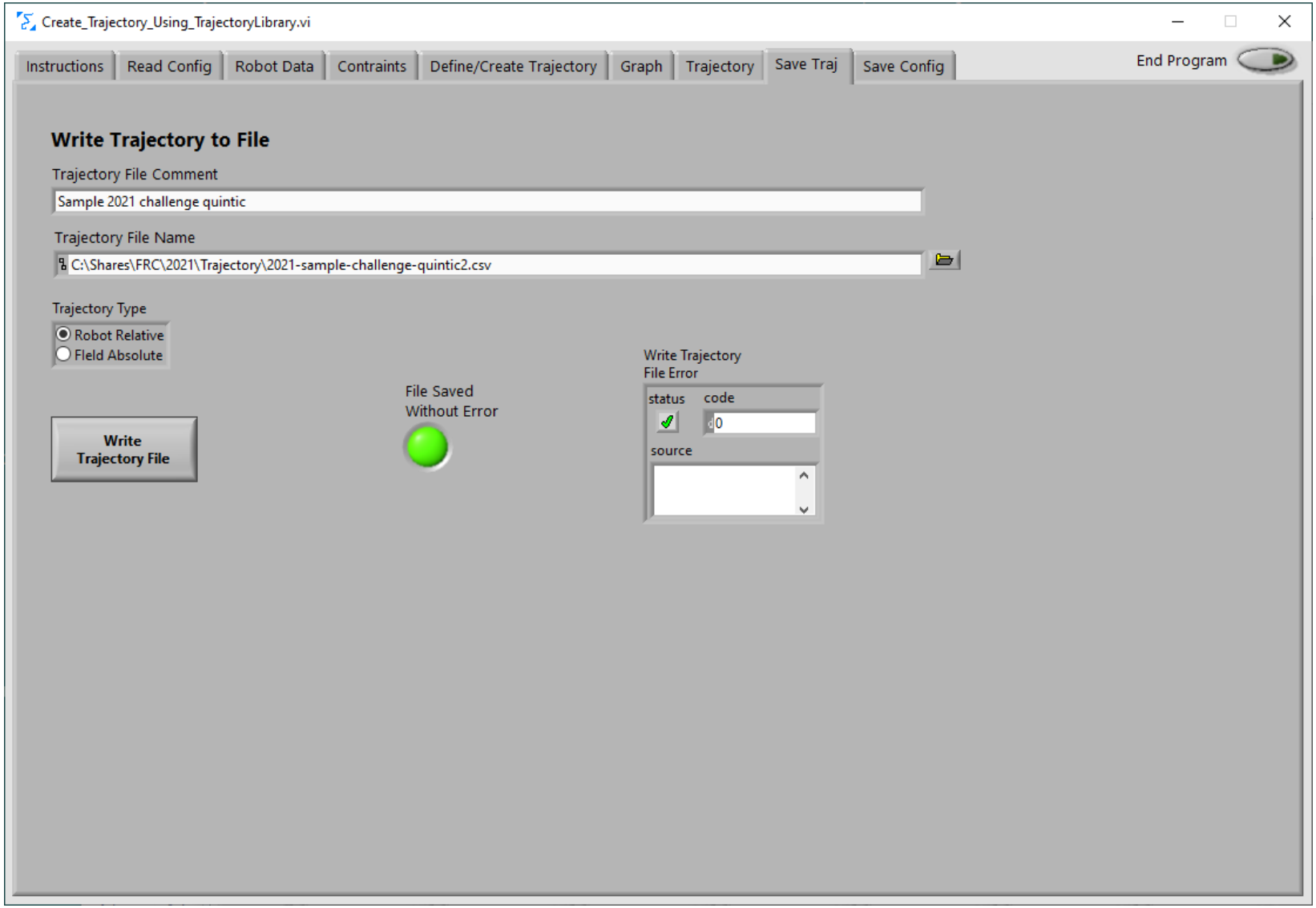
Trajectory Creation – Review Trajectory



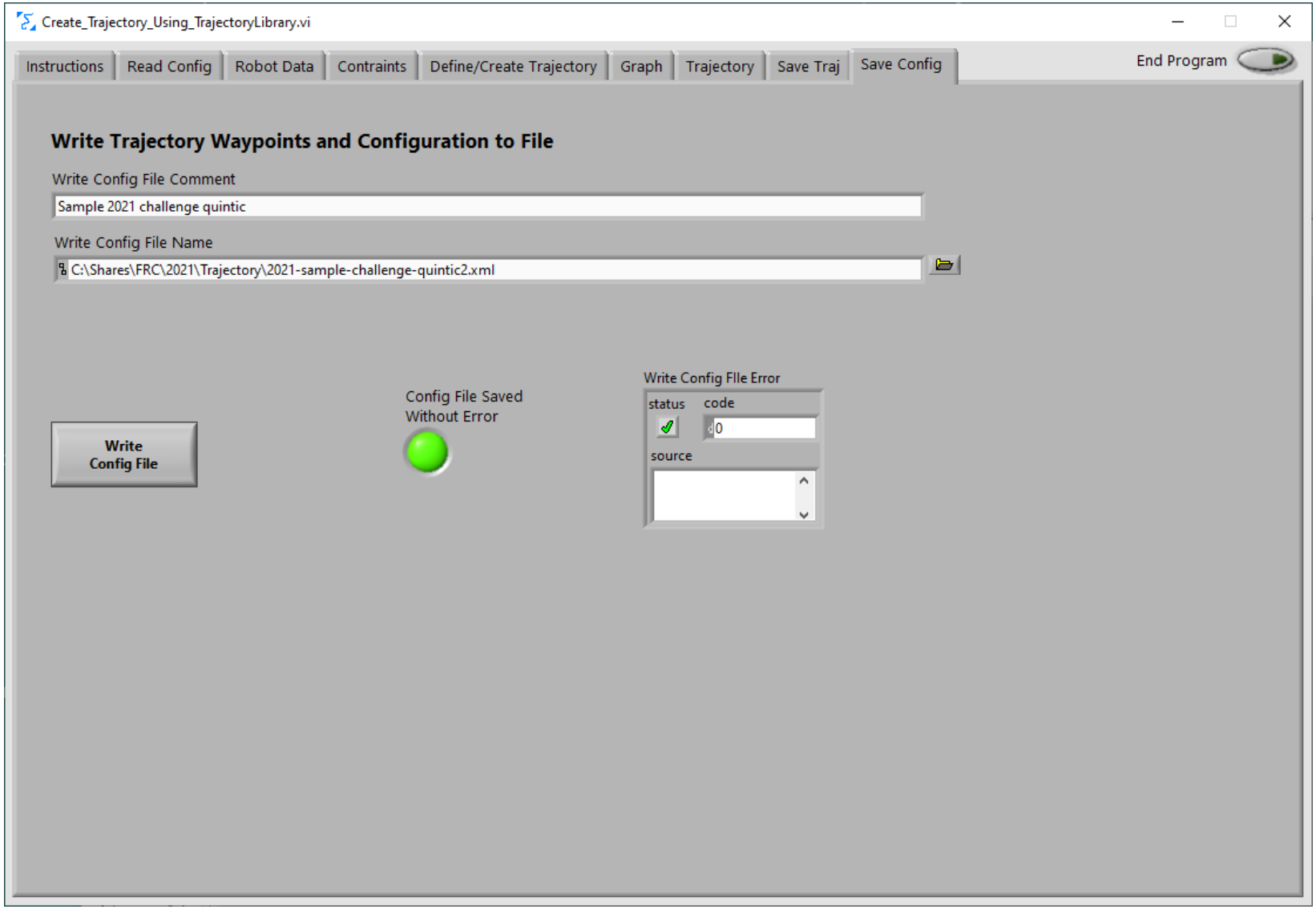
Trajectory Creation – Review Trajectory



Trajectory Creation – Write Trajectory



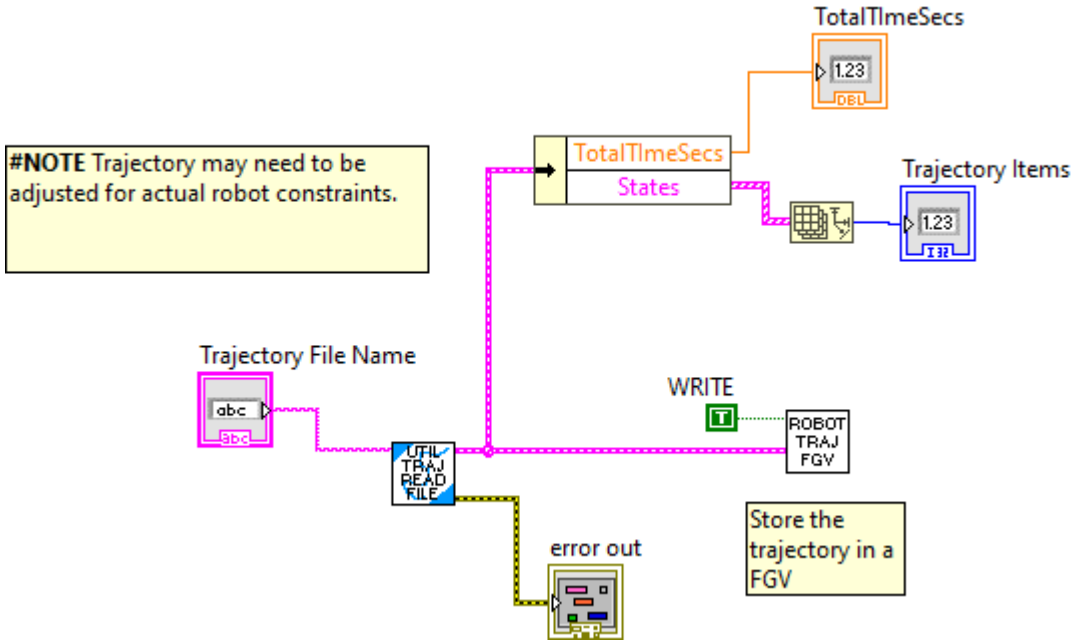
Trajectory Creation – Write configuration



Trajectory Following

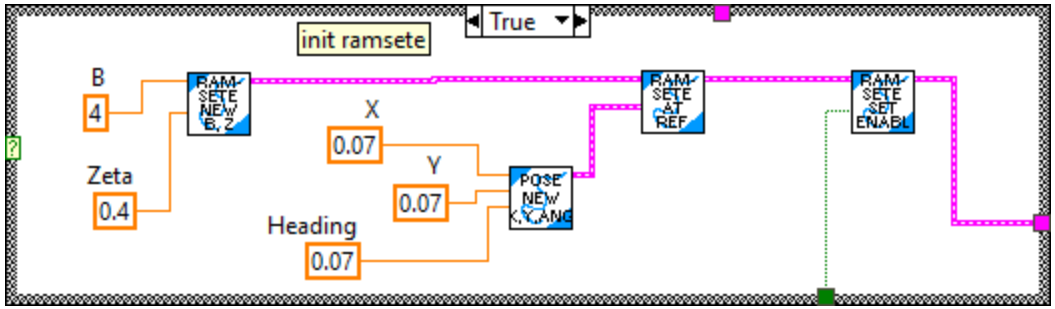
LabVIEW Code Sample

Sample – Read Trajectory File



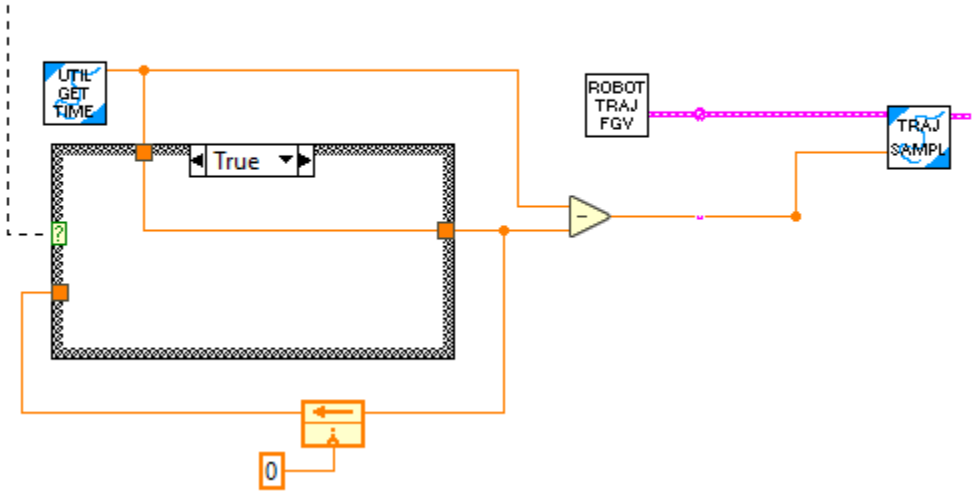
- Read trajectory file. Done once, when robot powered on.
- Populate memory data structure
- Report total time and trajectory states for reporting to dashboard via network tables.

Sample – Initialization Trajectory Execution



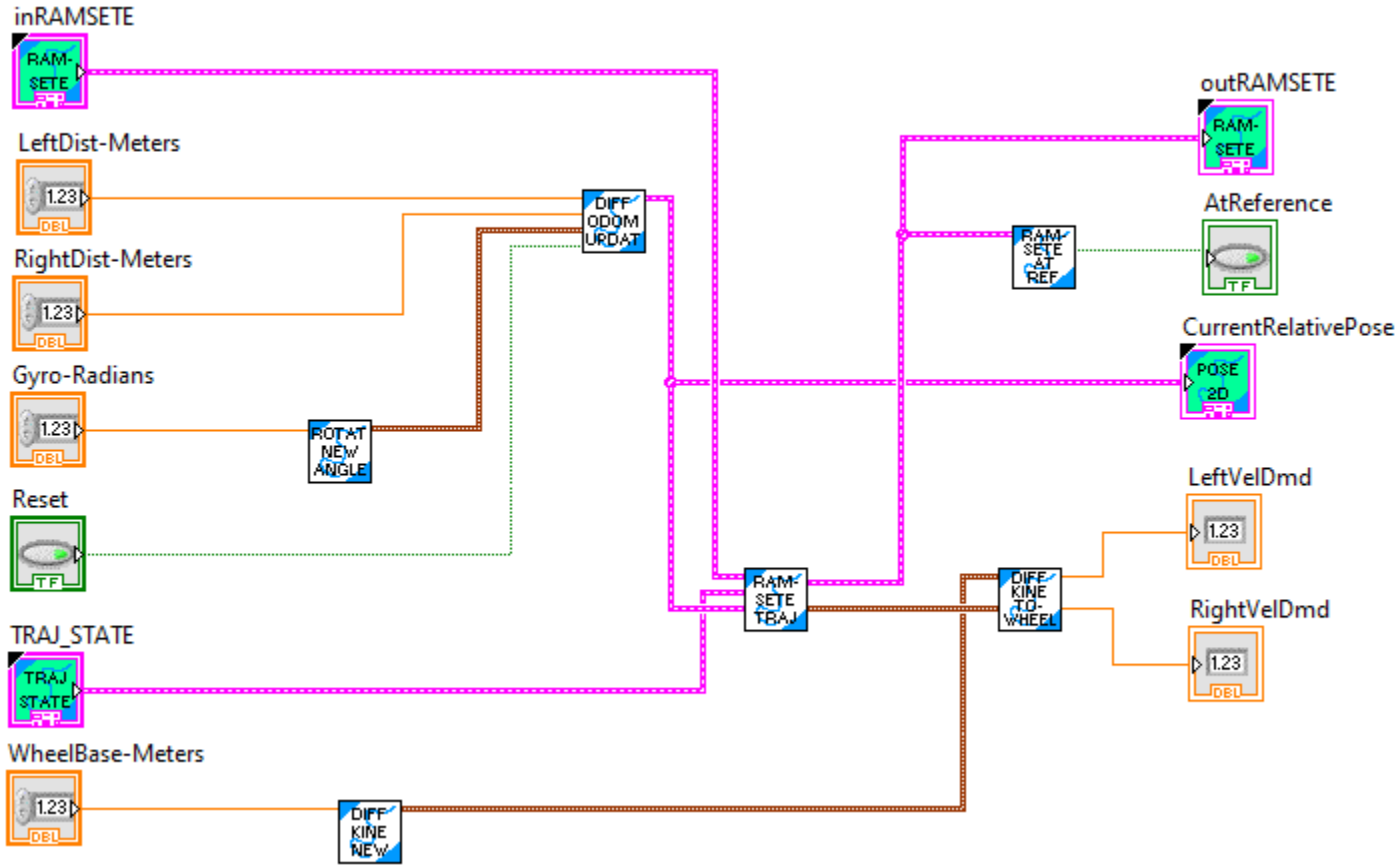
- Done once either at robot power up or before starting trajectory.
- Set Ramsete tuning
- Set allowed error to report “on target”.
- Enable ramsete control. During debugging this could be controlled from dashboard input.

Sample – Get trajectory sample



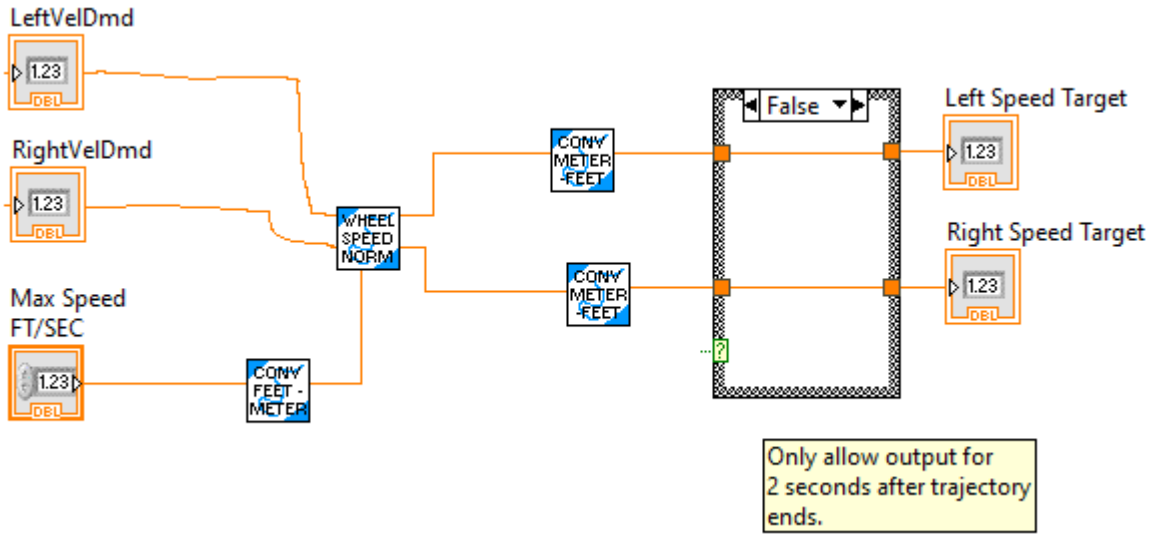
- Get time and calculate time offset since start of trajectory execution
- Get trajectory sample for the current time offset

Sample – Calc desired wheel speeds



- Update robot position (Odometry) using encoder distances, and gyro reading
- Using trajectory state and current position execute ramsete to calc desired speed. Then convert chassis speed to wheel speeds

Sample – Normalize wheel speeds



- Normalize wheel speeds. (Ensure speed doesn't exceed maximum.)
- Convert to units used by speed controller.
- If we are X seconds past trajectory execution force wheel demands to 0.