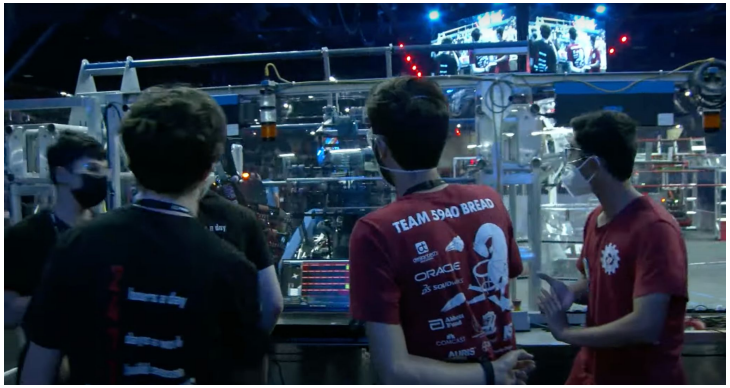
Design for Autonomous

Griffin Della Grotte

Special Thanks to Gautam and Krish from 5940

About Me

- Advisor, Technical Mentor, and Drive Coach for 5940
- System Engineer with Apple's Satellite Connectivity Group (SOS via Satellite)
- Security testing enthusiast
- Ex Race Car driver
- New cat owner!



Einstein 2022

Behind the fence-line of some datacenter

Lovely kitty

Championship FSAE autocross drive 2018

About this slide deck

Walk through the process of going from nothing to a high-performance autonomous mode.

In other words: there's more to a good auto than fancy code.

Team Insights

Number	Name	EPA Rank	Normalized EPA	EPA	Auto EPA	Teleop EPA	Endgame EPA	Record
6328	Mechanical Advantage	110	1721	55.5	22.4	27.9	5.2	47-23-1
1323	MadTown Robotics	1	2064	86.7	21.8	55.2	9.8	52-1-0
2468	Team Appreciate	12	1840	66.3	21.6	37.1	7.7	53-9-3
5940	BREAD	4	1937	75.1	20.7	45.6	8.9	40-9-0
1771	North Gwinnett Robotics	120	1716	55.1	19.8	28	7.4	46-15-0
3005	RoboChargers	5	1907	72.4	19.7	45.6	7.1	58-12-0
111	WildStang	6	1893	71.1	19.7	42.5	8.9	38-5-1
1325	Inverse Paradox	23	1818	64.4	19.7	34.9	9.8	60-26-0
4678	CyberCavs	13	1834	65.8	19.5	39.5	6.8	46-8-0
2767	Stryke Force	18	1824	64.9	19.3	37.7	8	52-14-0

Game release

How should we think about the autonomous mode? What advantage can we gain?

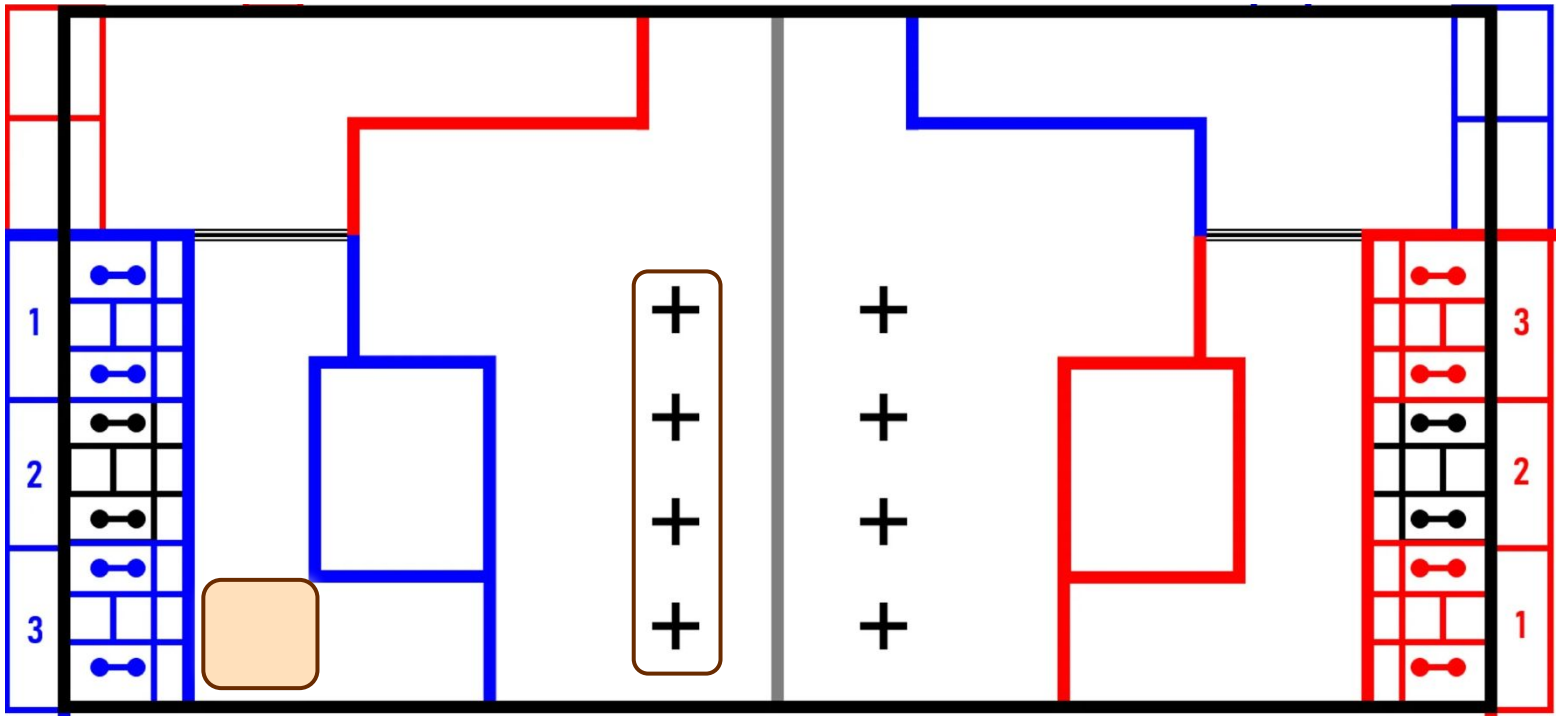
- Bonus points available only during auto
  - Remember it's about the delta (mostly)
- Getting an early lead
- Are there more?

Table 6-2 CHARGED UP points

Award	Awarded for…	AUTO	TELEOP	Qual.	Playoff
MOBILITY	each ROBOT whose BUMPERS have completely left its COMMUNITY at any point during AUTO	3			
GAME PIECES	scored on a bottom ROW	3	2		
	scored on a middle ROW	4	3		
	scored on a top ROW	6	5		
LINK	3 adjacent NODES in a ROW contain scored GAME PIECES.		5		
DOCKED and not ENGAGED	Each ROBOT (1 ROBOT max in AUTO)	8	6		
DOCKED and ENGAGED	Each ROBOT (1 ROBOT max in AUTO)	12	10		
PARK	Each ROBOT whose BUMPERS are completely contained within its COMMUNITY but does not meet the criteria for DOCKED.		2		
SUPERCHARGED NODE	each SUPERCHARGED NODE in a completed set of ALLIANCE GRIDS		3		
SUSTAINABILITY BONUS	At least 6 LINKS scored.			1 Ranking Point	

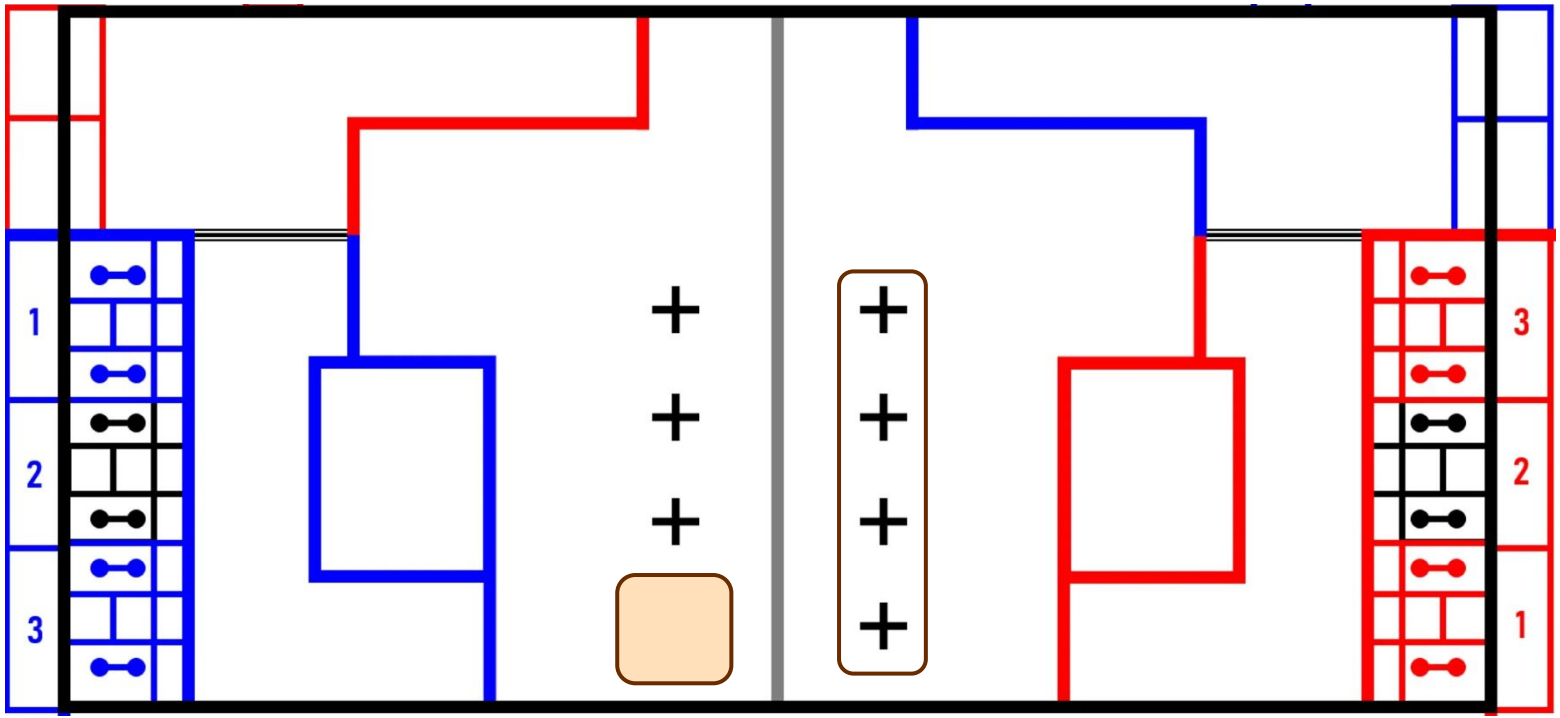
Game release

Short cycle for you, removal of short cycle for your opponent



Game release

Good field positioning going into teleop



Game release

Get a lead in the tiebreakers

Without new rules, champs 2023 *would have been decided on fouls or auto*

Order Sort	Criteria
1st	Ranking Score
2nd	Average ALLIANCE MATCH points, not including FOULS
3rd	Average ALLIANCE CHARGE STATION points
4th	Average ALLIANCE AUTO points
5th	Random sorting by the FMS

Unlikely to matter

2023 Qualification Ranking

Order Sort	Criteria
1st	Cumulative TECH FOUL points due to opponent rule violations
2nd	ALLIANCE CHARGE STATION points
3rd	ALLIANCE AUTO points
4th	MATCH is replayed

Clean match = TIE

Full-grid + Triple = TIE

Better auto???

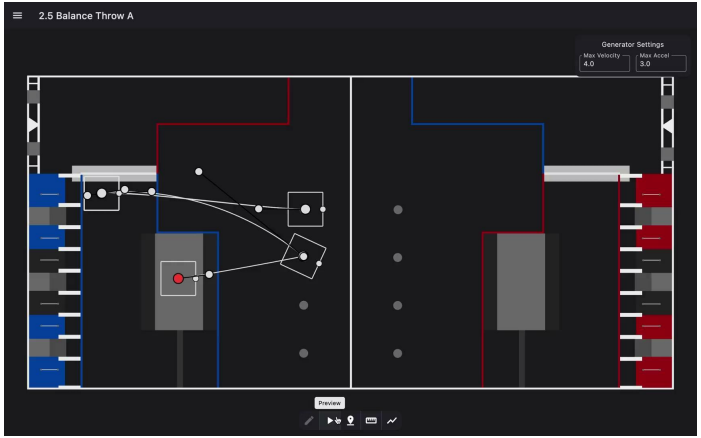
2023 Playoff Tiebreaker

Remember the pre-champs rules!!

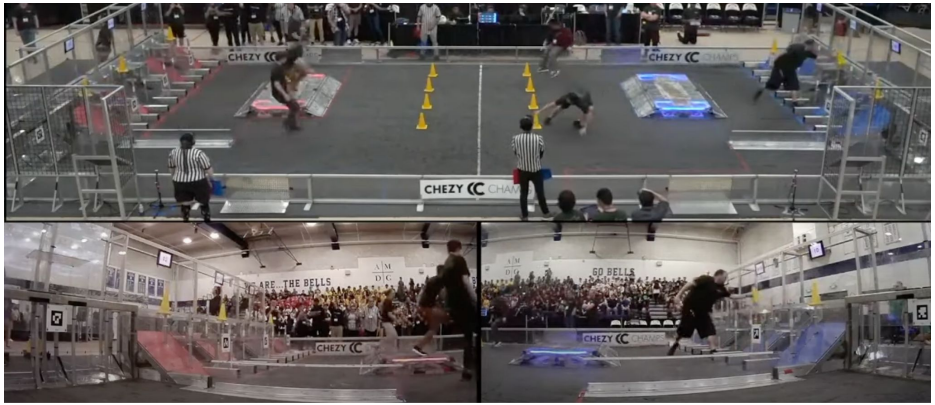
Field sketches + time estimates

Estimate how much is feasible to do in those 15 seconds. When in doubt, play it out!

This should let you know what the most *anyone* could do in auto will be.



Auto path in Path Planner



2023 Chezy Champs “Human Match”

Down-select your options

Which feasible auto routines do we want the most? Which don't matter much? What do you need to best serve your alliance?

Quals

- Most matches, not much help
- Maximize score with partners on the field
- Reasonable to work around your partners

Playoffs

- Playing with another very good robot
- Who does what? How do these modes complement each other?

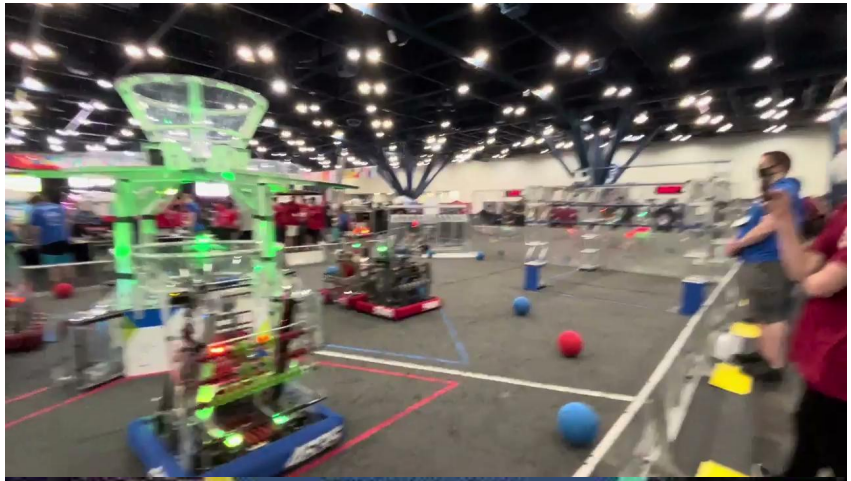
Down-select your options (2023)

1. **3 Balance**: Help enough to reach RP threshold, other partners can score a preload
2. **3 No Balance over Bump**: Playing with someone that has a good smooth side auto
3. **4 Toss + Balance**: No help, but also likely win, just trying to reach RP threshold
4. **3 No Balance**: Playing with a team that has only a center balance, or balance works well
5. **Only Preload**: Backup plan in case of mechanical issues or miscommunication
6. **2 Over Charge Station + Balance**: Playing with two very good teams

Remember, FRC is a game of *alliances*. The more flexible you are, the better you can make your whole alliance!

Down-select your options (2022)

1. **5 Piece Right**: Lots of points, leaves space for your alliance members
2. **2 Piece Left**: Secondary if playing with a team with a good 5 piece.
3. **3 Piece Steal Left**: Gotta get some extra points somewhere
4. **Only Preload**: Backup plan in case of mechanical issues or miscommunication
5. ~~**6 Piece**~~: Most points, but too hard on the alliance.



Consider your robot architecture

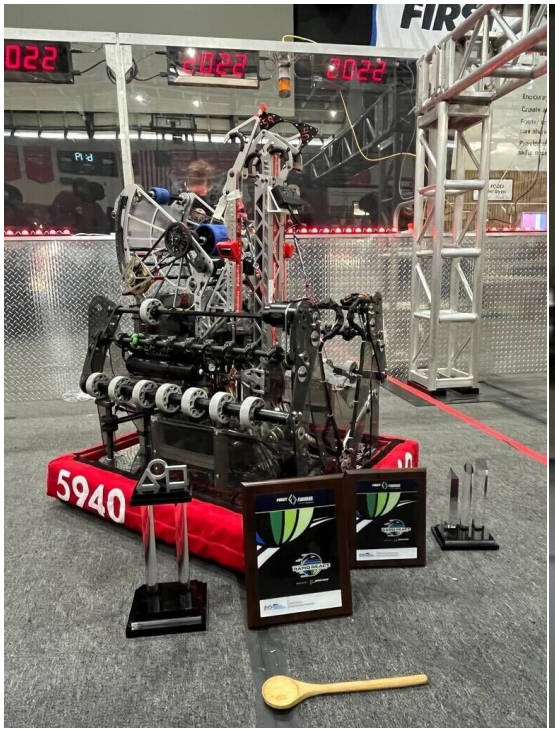
Certain architectures help or hurt your chances.

Consider your robot architecture

Certain architectures help or hurt your chances.

Consider your robot architecture

Certain architectures help or hurt your chances.



Consider your robot architecture

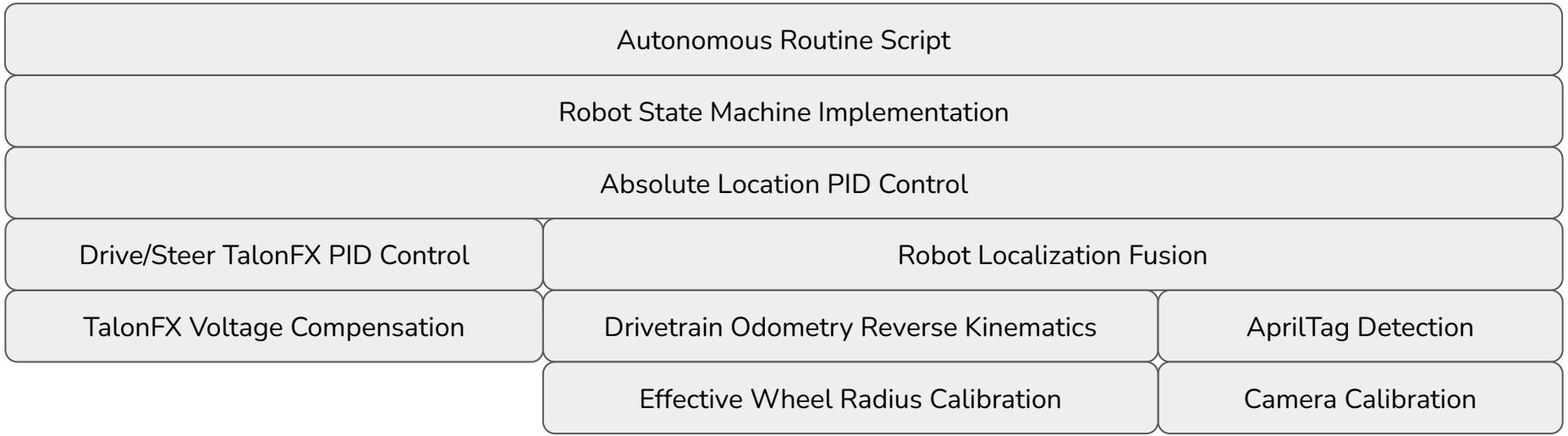
Certain architectures help or hurt your chances.

Consider your robot architecture

These choices might make your autos easier to achieve. Are the strictly necessary? Sometimes not, sometimes yes.

Implementation

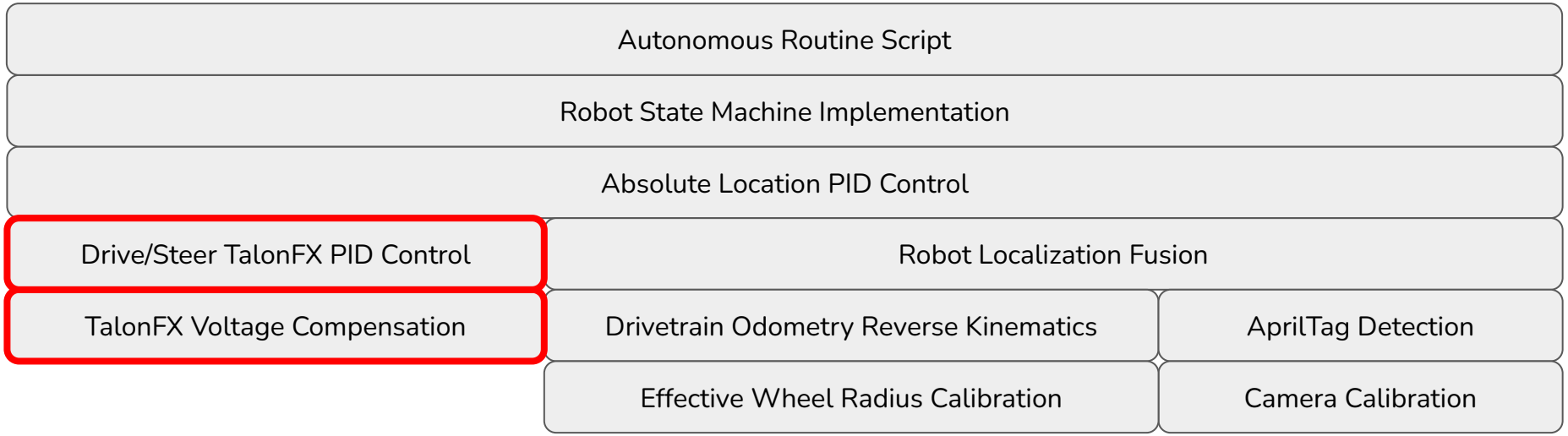
Layers of control



Many layers of automation to make a great auto happen!

Every layer needs to do its job...

Layers of control



Many layers of automation to make a great auto happen!

Every layer needs to do its job...

Motor control

Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation

Many places to find pre-tuned values for your swerve drive control loops. Other team's code, module supplier example code, CTRE's new Swerve wizard.

Worth checking when you get driving! Are the swerve modules actually achieving the setpoints they're given?

<u>Suggested tests</u>

- Check the commanded position closely matches the achieved position of the swerve rotation while driving.
- Run the robot in an automated forward path some distance. Check the achieved velocity of the modules closely matches the command.

Control Loop Options

Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation

Many different ways to implement PID control on your motors. A couple notes / recommendations:

On TalonFX
- 1000Hz PID for Pos. or Vel.
- Motion Profiling w/ Position PID

MotionMagic is 👍 and Phoenix 6 is 👍

On SPARK
- 1000Hz PID for Pos. or Vel.
- Motion Profiling w/ Velocity PID

PID control works! Beware low encoder count on NEOs for position. For motion profiling, use Rio + SPARK's position PID.

On Rio
- Main loop at 50Hz, can do ~faster thread
- Arbitrary PID / profile / model

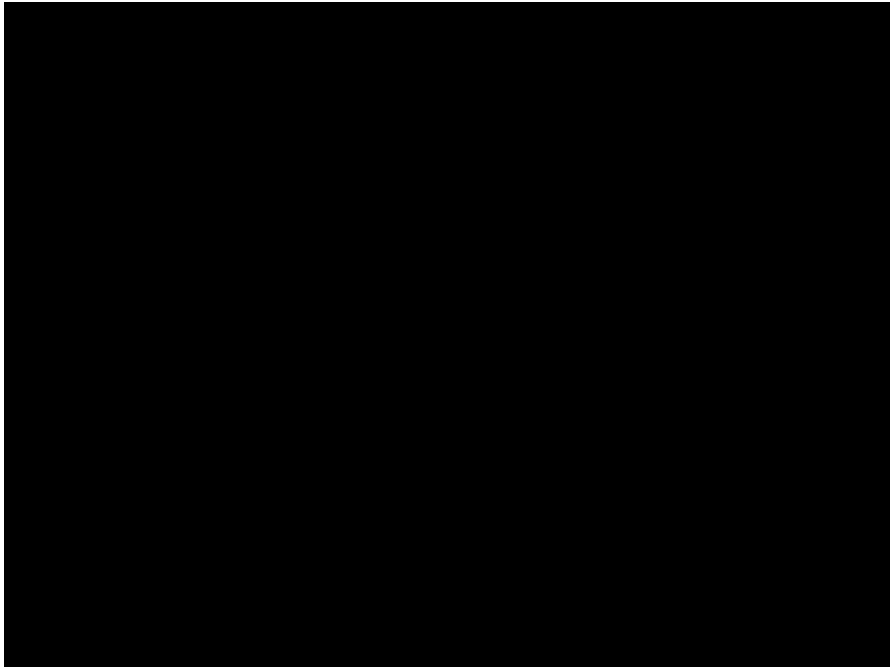
Avoid if you can. Better to avoid sensor latency and offload fast processing. Can often use higher gains with the faster loop speeds.

Control Loop Tuning

Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation

Depending on the performance you're looking for, this can be easy or hard. Seconds (or less) count!



Control Loop Tuning

Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation

kG: Add fixed voltage to offset gravity in one direction

kS: Add fixed voltage to offset static friction in the direction of rotation

kV: Add voltage proportional to the desired profile velocity

kA: Add voltage proportional to the desired profile acceleration

Feedforward

Output Voltage = kG + kS * sign(v) + kV * v + kA * a + kP * err + kI * accum-err-sec + kD * err/sec

kP: Add voltage proportional to the closed-loop error

kI: Add voltage proportional to the accumulated closed-loop error

kD: Add voltage proportional to the rate-of-change of the closed-loop error

Feedback

Control Loop Tuning

Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation

kG: Add fixed voltage to offset gravity in one direction

kS: Add fixed voltage to offset static friction in the direction of rotation

kV: Add voltage proportional to the desired profile velocity

kA: Add voltage proportional to the desired profile acceleration

kP: Add voltage proportional to the closed-loop error

kI: Add voltage proportional to the accumulated closed-loop error

kD: Add voltage proportional to the rate-of-change of the closed-loop error

GravityType		Gravity Feedforward Type
kA	Volts / rps/s	Acceleration Feedforward Gain
kD	Volts / Δ error-rotations	Derivative Gain
kG	Volts	Gravity Feedforward Gain
kI	Volts / accumulated-error-rotations	Integral Gain
kP	Volts / error-rotations	Proportional Gain
kS	Volts	Static Feedforward Gain
kV	Volts / rps	Velocity Feedforward Gain

Available PID contents in Phoenix 6
(Units for MotionMagicVoltage)

Control Loop Tuning (kG & kS)

Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation

1. Find your gravity offset (if needed, like for an arm)
   a. Increase kG until your mechanism supports its own weight without moving
   b. Increase kG more until the mechanism accelerates against gravity
   c. Center of that range is **kG**, half the width of that range is **kS**.
2. If no gravity offset is needed, can find kS on it's own in a similar way.

Starting point? Do a hand-calc to find the voltage needed to sustain the torque required.

Take care of the input and outputs on these functions. In Phoenix 6, we can choose a couple options. Recommend start with MotionMagicVoltage so the outputs are "voltage compensated" for variable robot battery.

`GravityType`		Gravity Feedforward Type
`kA`	Volts / rps/s	Acceleration Feedforward Gain
`kD`	Volts / Δ error-rotations	Derivative Gain
`kG`	Volts	Gravity Feedforward Gain
`kI`	Volts / accumulated-error-rotations	Integral Gain
`kP`	Volts / error-rotations	Proportional Gain
`kS`	Volts	Static Feedforward Gain
`kV`	Volts / rps	Velocity Feedforward Gain

Available PID contents in Phoenix 6
(Units for MotionMagicVoltage)

Control Loop Tuning (kV)

Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation

1. Find the velocity coefficient
    a. Adjust until the motion profile roughly matches the setpoint

The mechanism will not actually drive to its end position at this point, but when traversing the profile, it should be close!

Starting point? Direct translation from motor free speed per volt.

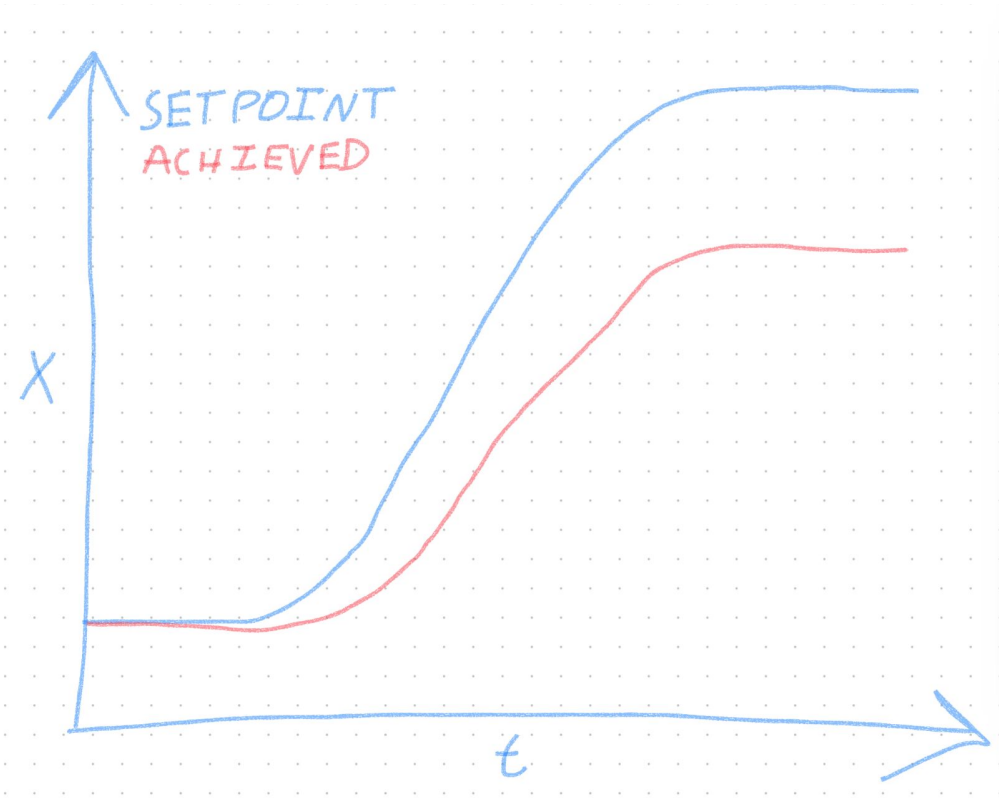
`GravityType`		Gravity Feedforward Type
`kA`	Volts / rps/s	Acceleration Feedforward Gain
`kD`	Volts / Δ error-rotations	Derivative Gain
`kG`	Volts	Gravity Feedforward Gain
`kI`	Volts / accumulated-error-rotations	Integral Gain
`kP`	Volts / error-rotations	Proportional Gain
`kS`	Volts	Static Feedforward Gain
`kV`	Volts / rps	Velocity Feedforward Gain

Available PID contents in Phoenix 6
(Units for MotionMagicVoltage)

Control Loop Tuning (kV)

Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation

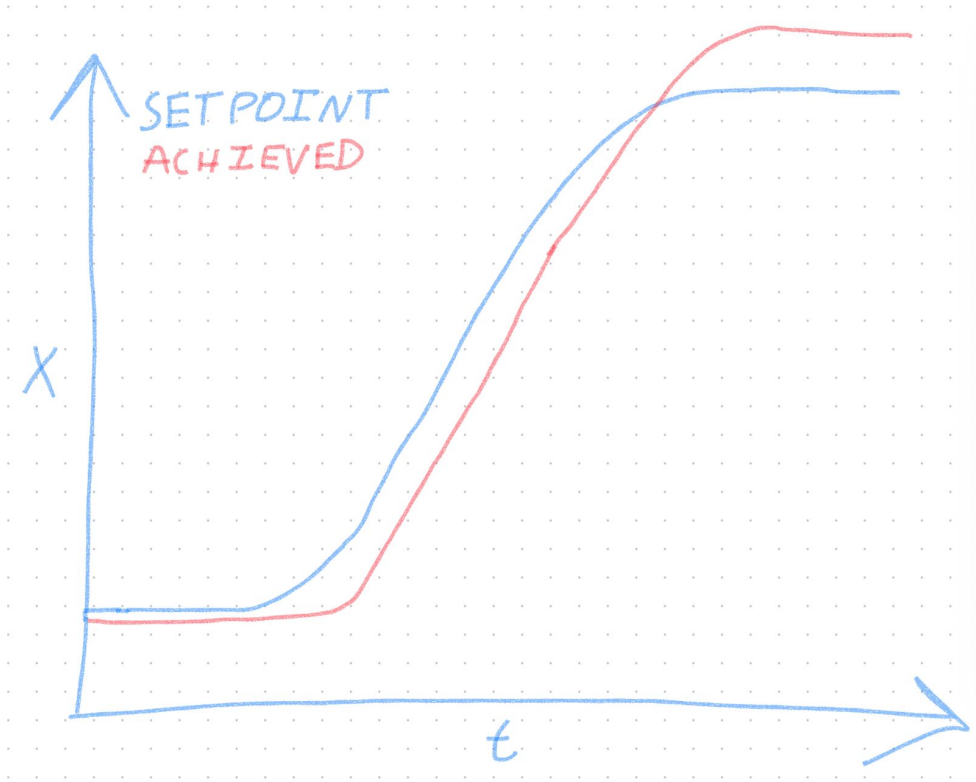


Looking to get the cruise portion (the linear increase portion) of the blue line and red line parallel.

Control Loop Tuning (kV)

Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation



Line lags behind blue until overshooting at the end, this is OK for now!

Control Loop Tuning (kA)

Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation

1. Find the acceleration coefficient (if needed)
    a. Add to the coefficient in order to help the mechanism accelerate and decelerate at the start and end of the profile

The mechanism will not actually drive to its end position at this point, but when traversing the profile, it should be close!

Many mechanisms will not need this. Only comes into play when you're really pushing limits. We implemented this manually last year.

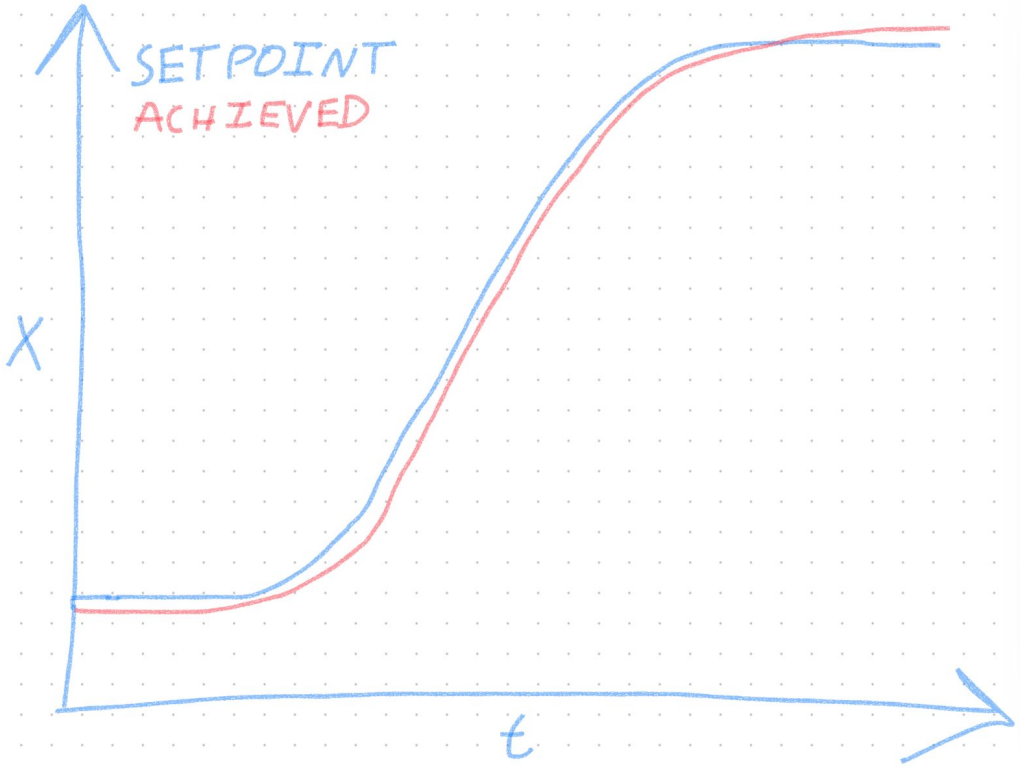
GravityType		Gravity Feedforward Type
kA	Volts / rps/s	Acceleration Feedforward Gain
kD	Volts / Δ error-rotations	Derivative Gain
kG	Volts	Gravity Feedforward Gain
kI	Volts / accumulated-error-rotations	Integral Gain
kP	Volts / error-rotations	Proportional Gain
kS	Volts	Static Feedforward Gain
kV	Volts / rps	Velocity Feedforward Gain

Available PID contents in Phoenix 6
(Units for MotionMagicVoltage)

Control Loop Tuning (kA)

Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation



kA brings overshoot and lag into check, but some position error remains

Control Loop Tuning (kP)

Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation

1. Find the proportional gain
    a. Add more P until you see oscillations, back off.
    b. The control will already look quite good if you did the previous steps well!

Motion profiled control loops can handle higher P terms because the error values ever presented should be smaller.

Golden rule of controls: If *you* know something about the system, tell the controller!

This is the “magic” of motion profiles. The mechanism cannot accelerate instantaneously, or go arbitrarily fast.

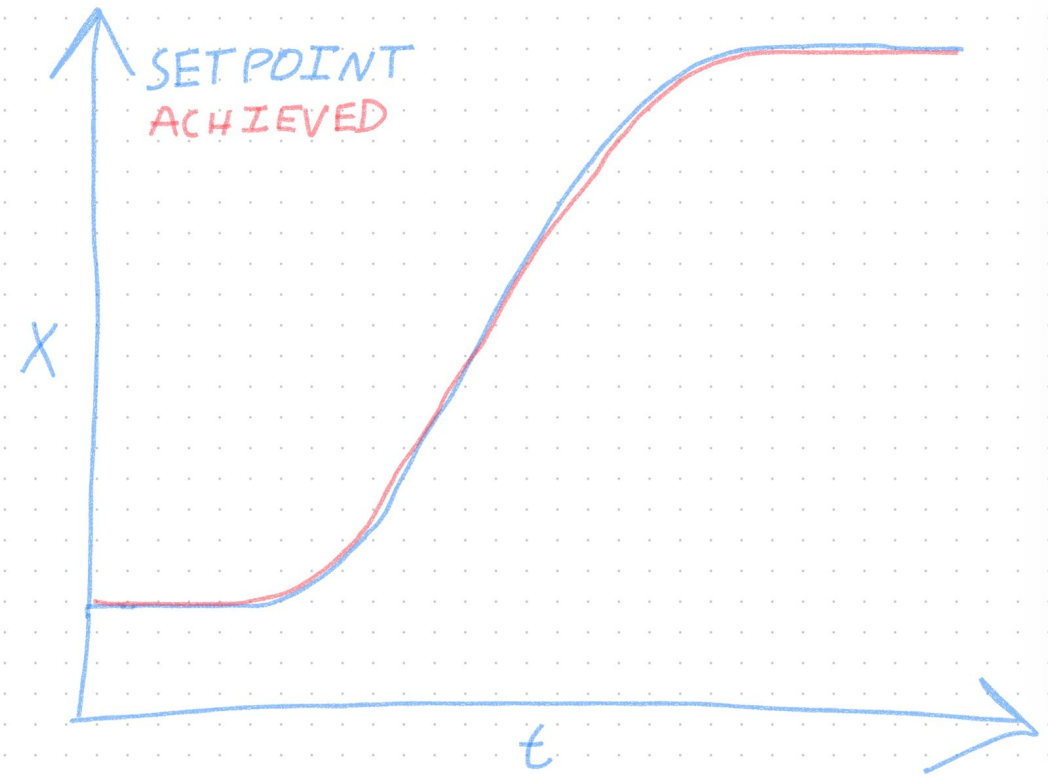
GravityType		Gravity Feedforward Type
kA	Volts / rps/s	Acceleration Feedforward Gain
kD	Volts / Δ error-rotations	Derivative Gain
kG	Volts	Gravity Feedforward Gain
kI	Volts / accumulated-error-rotations	Integral Gain
kP	Volts / error-rotations	Proportional Gain
kS	Volts	Static Feedforward Gain
kV	Volts / rps	Velocity Feedforward Gain

Available PID contents in Phoenix 6
(Units for MotionMagicVoltage)

Control Loop Tuning (kP)

Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation



kP handles remaining position error

Control Loop Tuning (kI & kD)

Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation

1. Find the I and D terms
    a. Set them to zero. Done 🙂

Haven't used a non-zero I term on an FRC mechanism for as long as I can remember. We use D sometimes, but it's usually for non-profiled PID controllers.

Our swerve module drive motors have a kD that is two orders of magnitude lower than the kP. Steer motors are running no kD.

GravityType		Gravity Feedforward Type
kA	Volts / rps/s	Acceleration Feedforward Gain
kD	Volts / Δ error-rotations	Derivative Gain
kG	Volts	Gravity Feedforward Gain
kI	Volts / accumulated-error-rotations	Integral Gain
kP	Volts / error-rotations	Proportional Gain
kS	Volts	Static Feedforward Gain
kV	Volts / rps	Velocity Feedforward Gain

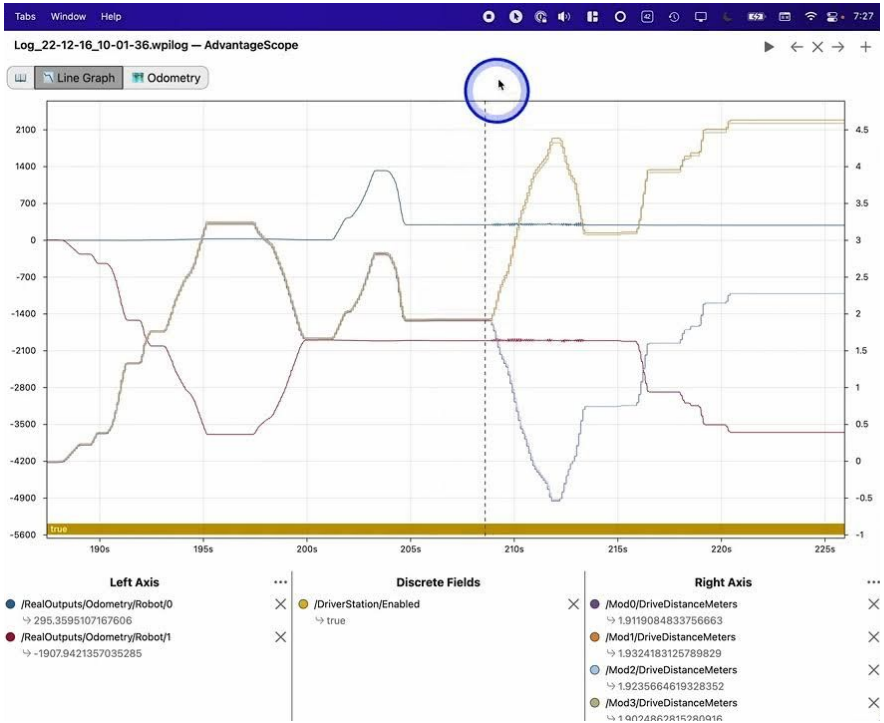
Available PID contents in Phoenix 6
(Units for MotionMagicVoltage)

Control Loop Tuning

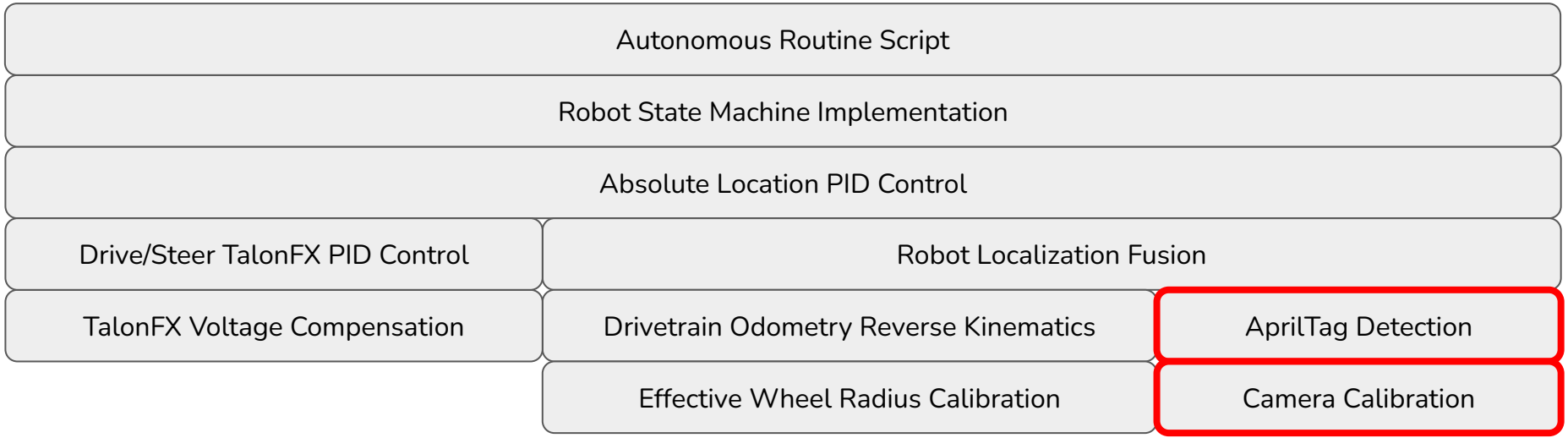
Drive/Steer TalonFX PID Control

TalonFX Voltage Compensation

All of this is done visually, plotting achievement vs. setpoint



Layers of control



Cameras

AprilTag Detection

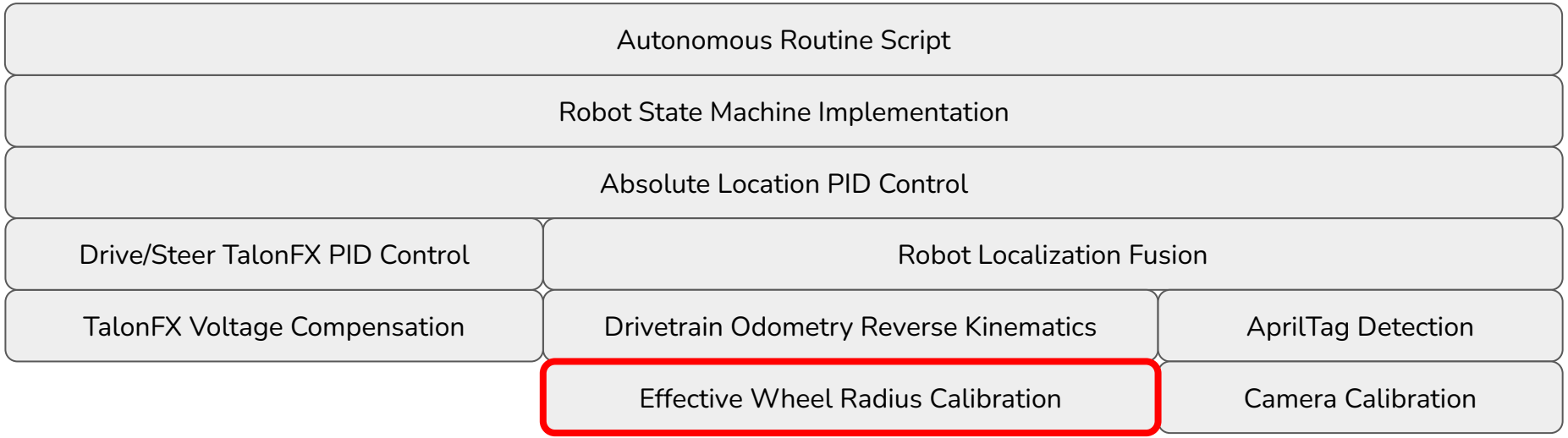
Camera Calibration

Lens calibration and positional calibration can be easy to miss. Lens first!

Positional calibration

- Start with the true offsets
- Try close to your tags and far from your tags
- Verify the detected camera position (and robot position by proxy) is correct. Adjust if not.
- Camera angle has a large effect only at far distances.
- Bad lens cal → wacky pos. cal

Layers of control



Carpet Calibration

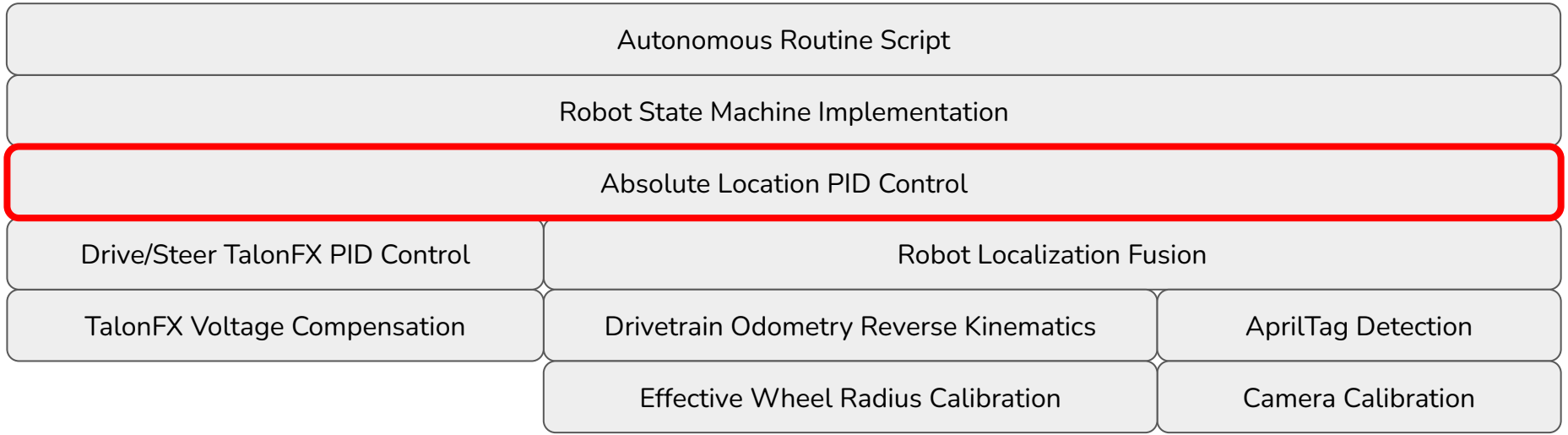
Effective Wheel Radius Calibration

Determine what the rolling radius of your wheels are

Test Procedure

- Autonomously drive a fixed distance *at low acceleration*
- Record the distance travelled reported by the sensors
- Measure the true distance travelled
- Calculate rolling radius of wheels

Layers of control



Path Follower PID

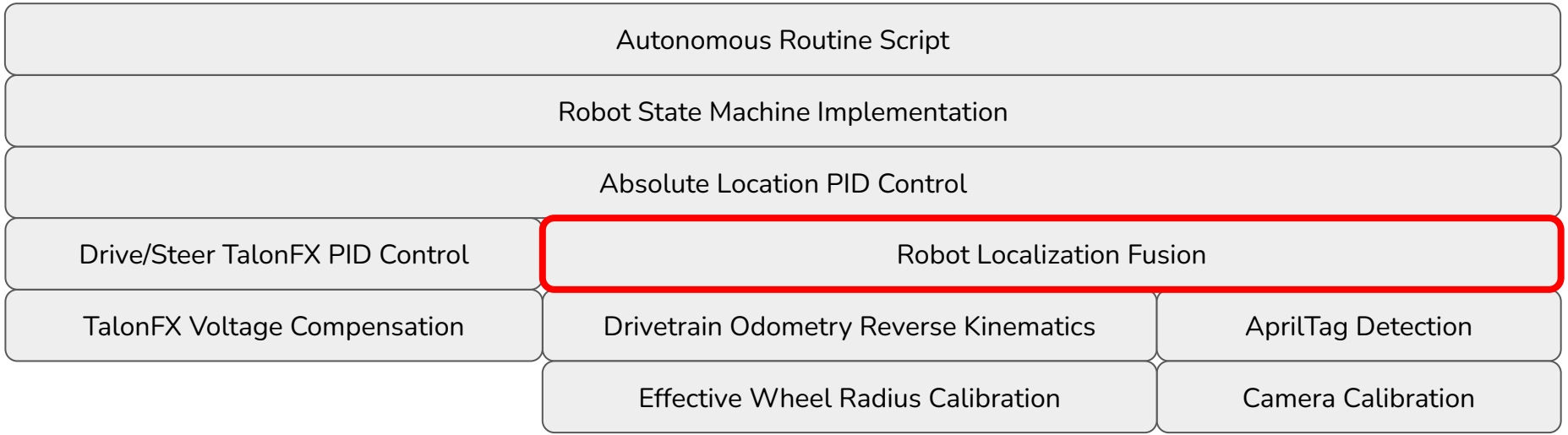
Path following usually implemented with a separate PID controller for X, Y, and rotation, yielding velocity commands for the swerve system.

We usually just give it as high a P term as can be sustained without inducing oscillations. Look to other team's code base for their choice!

Test Procedure

- Increase P term until jitter occurs during path following
- Do this *before* adding in vision to the robot localization, as that can also introduce oscillations

Layers of control



Sensor Fusion

Robot Localization Fusion

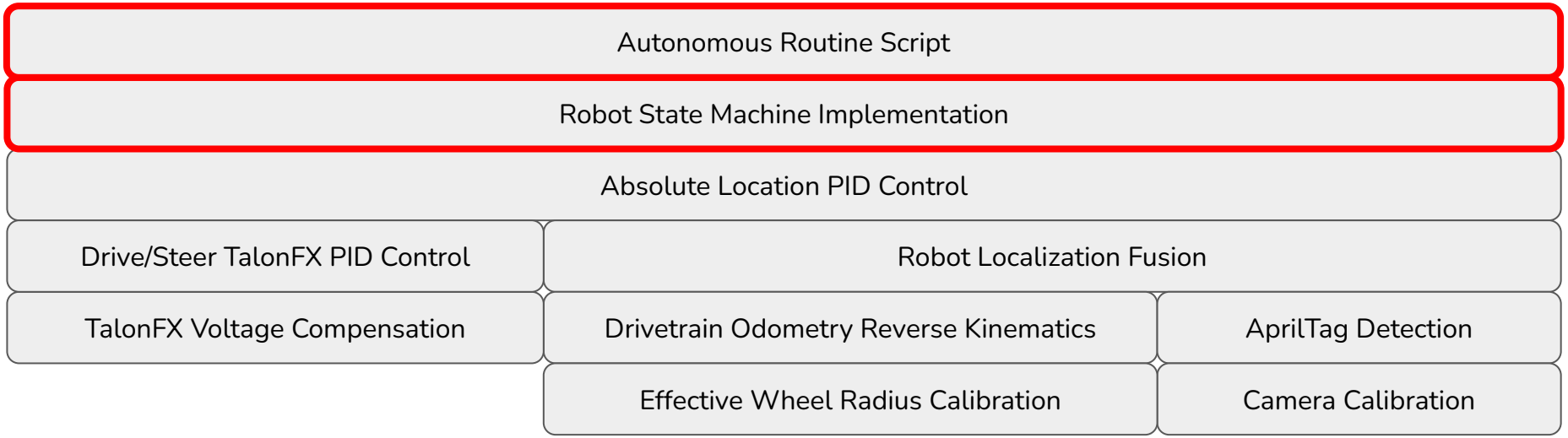
Tune how the robot handles combining swerve odometry data with camera measurements. Really just tuning which data is “trusted” more.

Can do more scientific measurements on the actual StdDev in your vision measurements, but we got better performance just tweaking it, so we’ll probably skip that part completely now.

Test Procedure

- Observe the “jitter” in your pose estimation. More trust in vision will lead to more jitter. Less trust will lead to less accuracy.
- Increase your trust in your vision measurements until the jitter becomes unacceptable for path following

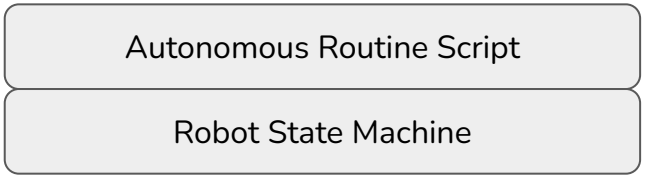
Layers of control



“Application Code”

Many different ways to script the robot actions that take place during an autonomous routine, see what works best for you.

This isn’t “performance” code. You can write the same auto many different ways at this point. Looking for something that is reasonable to read, and reasonable to tweak.



State Machine

Complex robot procedures defined through sequence of states.

“Signals” provide input to how the driver or autonomous routine wants the robot to change action

Can only transition from certain states into other states

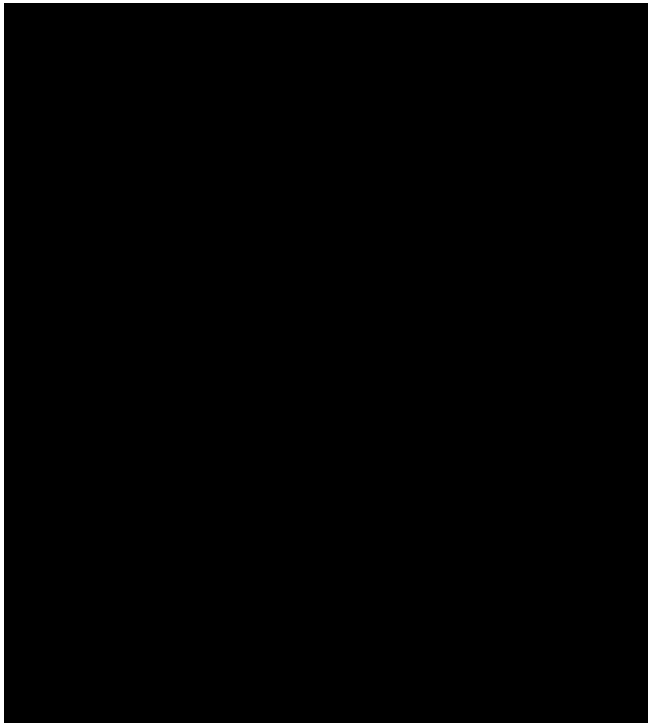
```
/* System States */
public enum SuperstructureState {
    PRE_HOME,
    HOMING,
    IDLE,
    PREPARE_TO_THROW,
    THROWING,
    FLOOR_INTAKE_CUBE,
    SINGLE_SUBSTATION_CONE,
    HP_INTAKE_CONE,
    HP_INTAKE_CONE_INTER,
    SPIT_CUBE_FRONT,
    PREPARE_TO_SPIT,
    SPIT,
    EXIT_SPIT,
    PRE_PLACE_PIECE_LOW,
    PRE_PLACE_CUBE,
    PRE_PLACE_CONE,
    EXHAUSTING_PIECE_LOW,
    EXHAUSTING_CUBE,
    SLAM_CONE,
    PULL_OUT_CONE,
    FLOOR_INTAKE_CONE_A,
    FLOOR_INTAKE_CONE_B,
    FLOOR_INTAKE_CONE_C
}
```

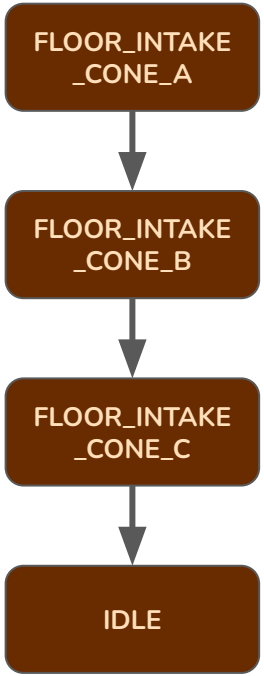
Robot State Machine

```
/** System states */
public enum ElevatorArmSystemStates {
    STARTING_CONFIG,
    NEUTRALIZING_ARM,
    HOMING,
    IDLE,
    FOLLOWING_SETPOINT
}

public enum FloorIntakeStates {
    IDLE,
    CLOSED_LOOP

}
```

State Machine

Robot State Machine





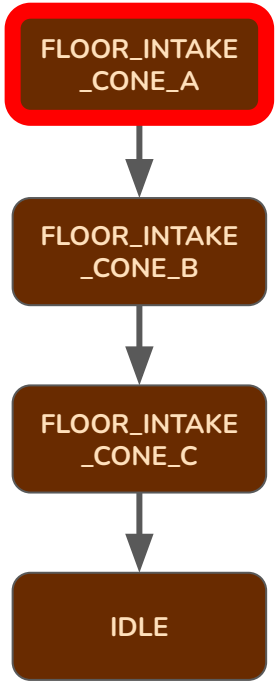
State Machine

Robot State Machine

```
} else if (systemState == SuperstructureState.FLOOR_INTAKE_CONE_A) {
    // Outputs
    endEffector.idling();
    if (floorIntake.getRollerCurrent() > 55.0) {
        floorIntake.requestClosedLoop(-0.75, -2.373046875);
    } else {
        floorIntake.requestClosedLoop(-0.75, 154.0);
    }
    elevatorArmLowLevel.requestDesiredState(ELEVATOR_IDLE_POSE, ARM_IDLE_POSE, goSlow);

    // Transitions
    if (!requestFloorIntakeCone) {
        nextSystemState = SuperstructureState.IDLE;
    } else if (floorIntake.getAngle() < 120.0 && floorIntake.getRollerRPM() > -500.0 &&
        nextSystemState = SuperstructureState.FLOOR_INTAKE_CONE_B;
    }
```

Start lifting the intake if the current suggests a cone is in it

Transition once the intake is lifted and roller jammed by a cone



State Machine

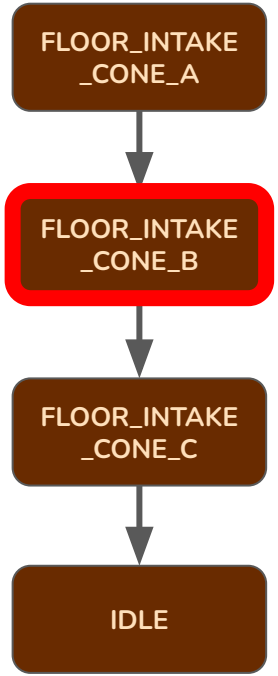
Robot State Machine

Extend elevator out, rotate arm in

```
} else if (systemState == SuperstructureState.FLOOR_INTAKE_CONE_B) {
    // Outputs
    if (elevatorArmLowLevel.getState()[0] > 0.4) {
        elevatorArmLowLevel.requestDesiredState(0.5, 24.67578125, goSlow);
    } else {
        elevatorArmLowLevel.requestDesiredState(0.5, 90.0, goSlow);
    }
    floorIntake.requestClosedLoop(-0.75, -2.373046875);
    endEffector.intakeCone();

    // Transitions
    if (elevatorArmLowLevel.atArmSetpoint(24.67578125) && elevatorArmLowLev
        nextSystemState = SuperstructureState.FLOOR_INTAKE_CONE_C;
    } else if (!requestFloorIntakeCone) {
        nextSystemState = SuperstructureState.IDLE;
    }
```

Different arm rotation depending on the position of the elevator

Transition once elevator + arm reaches end of travel



State Machine

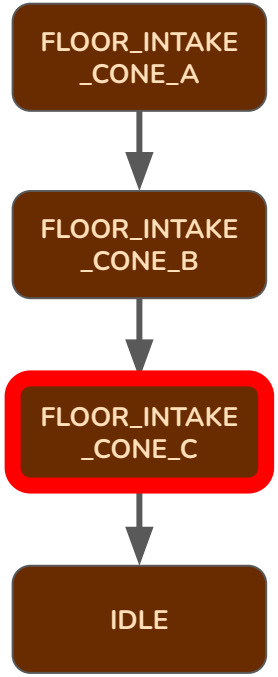
Robot State Machine

```
} else if (systemState == SuperstructureState.FLOOR_INTAKE_CONE_C) {
    // Outputs
    elevatorArmLowLevel.requestDesiredState(0.15, 24.67578125, goSlow);
    if (elevatorArmLowLevel.atElevatorSetpoint(0.15)) {
        floorIntake.requestClosedLoop(0.15, 0.0);
    } else {
        floorIntake.requestClosedLoop(-0.75, -2.373046875);
    }
    endEffector.intakeCone();


    if (endEffector.getBeamBreakTriggered() && !beamBreakTriggerTimeStarted) {
        beamBreakTriggerTimeStarted = true;
        beamBreakTriggerTimer.reset();
        beamBreakTriggerTimer.start();
    }

    if (!endEffector.getBeamBreakTriggered()) {
        beamBreakTriggerTimer.reset();
        beamBreakTriggerTimer.stop();
        beamBreakTriggerTimeStarted = false;
    }

    // Transitions
    if (!requestFloorIntakeCone) {
        nextSystemState = SuperstructureState.IDLE;
    } else if (beamBreakTriggerTimer.get() > 0.1) {
        nextSystemState = SuperstructureState.IDLE;
        requestFloorIntakeCone = false;
    }
```

Start spitting out the cone only once the arm is fully in position

Timing logic for how long the tusk beam-break has been triggered

Do not transition unless the tusks signal there is actually a cone in them



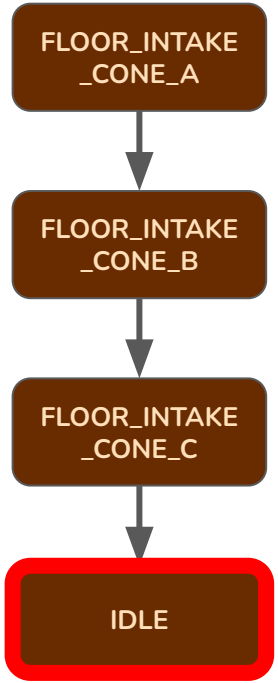
State Machine

Robot State Machine

```
} else if (systemState == SuperstructureState.IDLE) {
    // Outputs
    elevatorArmLowLevel.requestDesiredState(ELEVATOR_IDLE_POSE, ARM_IDLE_POSE, goSlow);
    if (elevatorArmLowLevel.atElevatorSetpoint(ELEVATOR_IDLE_POSE) && elevatorArmLowLeve
        if (endEffector.getBeamBreakTriggered()) {
            endEffector.holdCone();
        } else {
            endEffector.holdCube();
        }
    }
    floorIntake.requestClosedLoop(0.0, INTAKE_IDLE_POSITION);
    currentTriggerTimerStarted = false;

    // Transitions
    if (requestHome) {
        nextSystemState = SuperstructureState.PRE_HOME;
    } else if (requestFloorIntakeCube) {
        nextSystemState = SuperstructureState.FLOOR_INTAKE_CUBE;
    } else if (requestHPIntakeCone) {
```

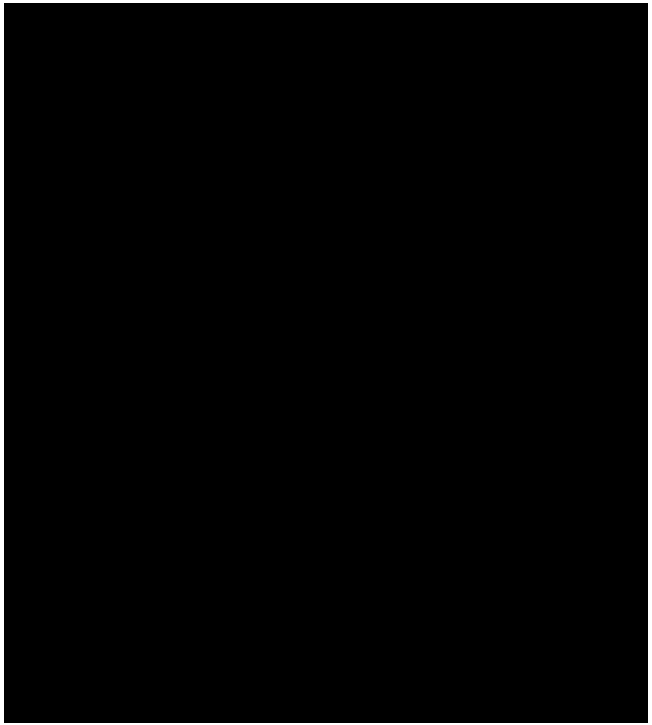
Different current limit for different gamepiece

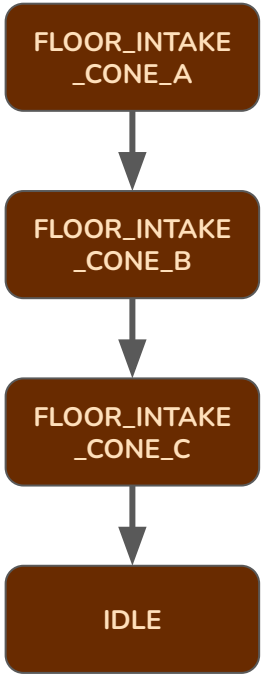
Big transition statement out of IDLE



State Machine

Robot State Machine





Routine Script

Autonomous Routine Script

```
public OnePieceBalanceMode(Swerve swerve, Superstructure superstructure) {
    addRequirements(swerve, superstructure);
    addCommands(
        new InstantCommand(() -> swerve.requestPercent(new ChassisSpeeds(0, 0, 0), false)),
        new InstantCommand(() -> superstructure.requestPreScore(Level.HIGH, GamePiece.CONE)),
        new WaitUntilCommand(() -> superstructure.atElevatorSetpoint(ELEVATOR_PRE_CONE_HIGH)),
        new InstantCommand(() -> superstructure.requestScore()),
        new WaitUntilCommand(() -> superstructure.atElevatorSetpoint(ELEVATOR_CONE_PULL_OUT_HIGH)),
        new WaitCommand(0.3),
        new InstantCommand(() -> superstructure.requestIdle()),
        new TrajectoryFollowerCommand(Robot.onePieceBalanceA, () -> Robot.twoPieceBalanceA.getInitial
        new WaitCommand(1.0),
        new TrajectoryFollowerCommand(Robot.onePieceBalanceB, swerve, true),
        new RunCommand(() -> swerve.requestPercent(new ChassisSpeeds(0, 0, 0), false))
    );
}
```

Routine Script

Autonomous Routine Script

```
public ThreePieceFloorConeMode(Superstructure superstructure, Swerve swerve) {
    addRequirements(superstructure, swerve);
    addCommands(
        new InstantCommand(() -> swerve.requestPercent(new ChassisSpeeds(0, 0, 0), false)),
        new InstantCommand(() -> superstructure.requestPreScore(Level.HIGH, GamePiece.CONE)),
        new WaitUntilCommand(() -> superstructure.atElevatorSetpoint(ELEVATOR_PRE_CONE_HIGH)),
        new InstantCommand(() -> superstructure.requestScore()),
        new WaitUntilCommand(() -> superstructure.atElevatorSetpoint(ELEVATOR_CONE_PULL_OUT_HIGH)),
        new WaitCommand(0.4),
        new TrajectoryFollowerCommand(Robot.threePieceFloorConeA, () -> Robot.threePieceSlowA.getInitialHolonomicPose()
            new RunCommand(() -> superstructure.requestFloorIntakeCone())
        ),
        new InstantCommand(() -> superstructure.requestFloorIntakeCone()),
        new TrajectoryFollowerCommand(Robot.threePieceFloorConeB, swerve, true),
        new AutoPlaceCommand(swerve, superstructure, () -> GamePiece.CONE, () -> Level.MID).until(() -> superstructure.
        new TrajectoryFollowerCommand(Robot.threePieceFloorConeC, swerve, true).raceWith(
            new RunCommand(() -> superstructure.requestFloorIntakeCube(() -> 0.0))
        ),
        new InstantCommand(() -> superstructure.requestFloorIntakeCube(() -> 1.0)),
        new TrajectoryFollowerCommand(Robot.threePieceSlowD, swerve, true).alongWith(new SequentialCommandGroup(
            new WaitCommand(1.5),
            new InstantCommand(() -> superstructure.requestPreScore(Level.HIGH, GamePiece.CUBE))
        )),
        new InstantCommand(() -> superstructure.requestScore()),
        new WaitCommand(0.5),
        new InstantCommand(() -> superstructure.requestIdle()),
        new WaitCommand(0.1),
        new TrajectoryFollowerCommand(Robot.threePieceSlowSetup, swerve, false),
        new RunCommand(() -> swerve.requestPercent(new ChassisSpeeds(0, 0, 0), false))
    );
}
```

Score a cone high

Intake a cone while driving out and back

Score a cone mid w/ vision

Intake a cube while driving out and back

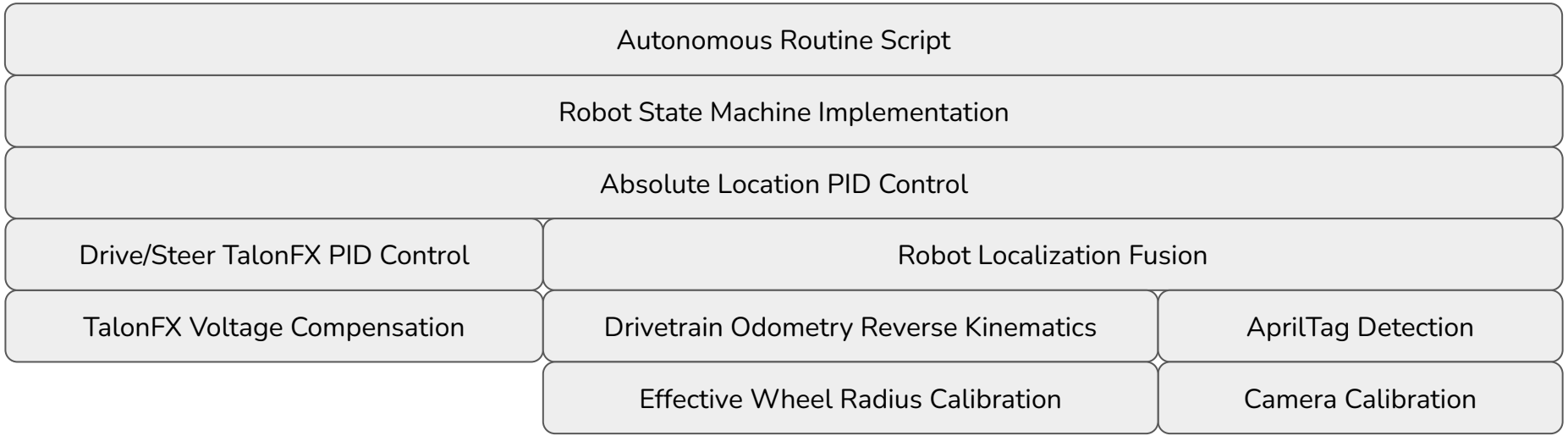
Score cube

Drive to midfield

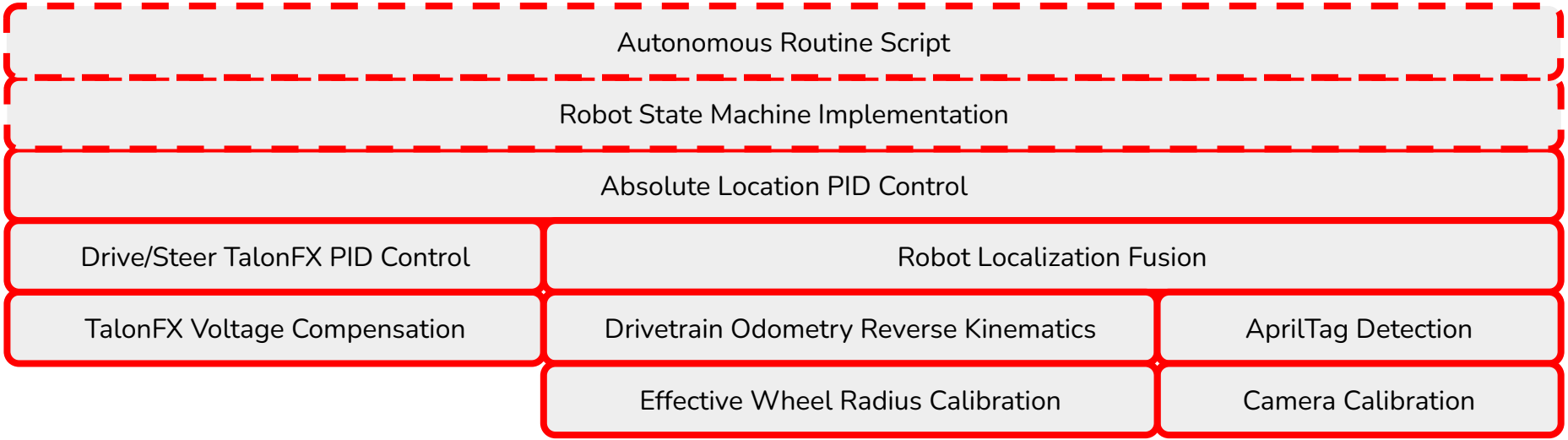
Routine Script

Autonomous Routine Script

Layers of control



Layers of control



Almost all of this can be done before the season starts!

Strategies for Success

- If it's possible to do it before the season, *do it before the season*.
- Think critically about what you *should* do in auto
- Resist the temptation to move higher in the layer stack.
- Tune graphically. You don't need a PhD to control a mechanism well.
- If it doesn't work every time in your shop, it probably won't work half the time on the field. Test unforgivingly.
- Be skeptical of hardware problems. Code rarely "just stops working" if it wasn't updated.

Discussion!