Flex + EZ Template

A Mentor's Guide to the Override Hero Bot Programming Curriculum

Use the V5 Flex Hero Bot as a second training robot — chassis PID, lift + claw control, sensor integration, seven driver-assist macros, and the path from kit-built to custom competition robot — all on the same robot students will see when the team's actual game robot arrives. One IMU and an omni-wheel swap convert stock Flex into a full EZ-Template platform.

This handout is a print-ready companion to the curriculum at **spartandesignrobotics.org/flex-training**. It's organized for mentors who want a single document on the bench while students work — section overviews, the key code that goes with each, hardware references, and notes on where students typically get stuck.

What	Detail
Robot	V5 Flex Hero Bot + V5 Inertial Sensor + omni-wheel swap
Framework	PROS kernel 4.1.x + EZ-Template 3.2.x
Curriculum length	8 sections, ~30 code blocks, ~25–30 hours of student work
Output	Working program: PID drive, lift + claw, 7 driver-assist macros, sensored auton
Audience	Drivers, Engineers, and Strategists — same robot, different lenses
Skill transfer	~85% of patterns carry directly to a custom competition robot

Contents

1. How to use this handout

Three audiences will pick this up. The structure below tells each one which sections are worth their time and which can be skimmed.

Mentors during sessions (most common)

You're at the bench with students. They're stuck. You need the answer fast. Use:

- **Section 5 (Section walkthroughs)** — find the section students are working on. The 'Key code' subsection is what they're trying to write. Show them; don't dictate.
- **Section 6 (Mentor implementation notes)** — five named traps. If a student's hitting one of them, this section names the problem and points to the fix.
- **Page 16 (Quick reference card)** — tear it off, tape it inside the toolbox lid. Common pitfalls, EZ-Template essentials, controller map.

Mentors before sessions (planning)

You're planning the next 2–3 sessions. You want to know what to budget time for and what hardware to prep. Use:

- **Section 2 (8 sections at a glance)** — hours per section. Section 6 (sensors) and Section 7 (macros) are the biggest time investments — budget two sessions each.
- **Section 3 (Hardware port map)** — what to prep for the day. Limit switches need ADI ports A and B; optical and distance need smart ports 3 and 7.

Lead mentors / coaches (program-level)

You're tracking the team's curriculum progress and using it to evaluate readiness for competition. Use:

- **Section 1 summary table** — what success looks like at the end of all 8 sections. ~85% skill transfer is the key number; it justifies the time investment.
- **Section 7 (Corrections log)** — track changes over time as the curriculum gets refined with use. Empty initially; fill in as you and the team find improvements.
- **Section 8 of the online guide** — the Flex→Custom comparison is the strongest Design Award notebook entry from the whole curriculum. Make sure students do it.

2. The 8 sections at a glance

Each section in the online curriculum follows the same pattern: framing, key code, exercises, common pitfalls. The hours column reflects student-work time, not mentor-prep time.

§	Section	What's taught	Hours
1	Why Flex is your second training robot	Pedagogy: why Flex vs. Clawbot, the four training tracks (build/driver/programming/notebook), what transfers between the two robots, the omni-wheel decision	~1
2	The sensors already on your Flex	IMU (port 13) verification with a 30-second print-loop test, motor encoders, what the bare robot can do vs. what it can't yet	~2
3	EZ-Template Flex configuration	subsystems.hpp port map, Drive constructor for 2-motor tank + 4" omni + green cartridge, initialize() PID seeds, verify-on-bench polarity flow	~3
4	Lift + claw control	Hold-to-move pattern for lift, toggle pattern for claw, current-limit stall protection (1.8A claw / 2.5A lift), brake-mode HOLD, time-based auton lift	~3
5	Progressive training exercises	Six staircased drills: drive 12in → 90° turn → 24in square → timed lift → drive-lift-score → two-block routine. Failure-as-data pattern.	~4
6	Sensors to add to your Flex	Five sensors with full integration code: bottom limit (zero-on-boot), top limit (safety), pot on lift pivot (closed-loop position), optical on claw, distance at back	~5–6
7	Driver-assist macros	Seven macros: position presets, smart grab, soft lift limits, precise turn, slow mode, rumble feedback, sequential score combo. Full opcontrol() orchestration with failsafes.	~5–6
8	From Flex to custom competition robot	What transfers (architecture, sensor patterns, macros, auton structure), what doesn't (PID constants, named positions, mechanism code). The Flex→Custom notebook comparison.	~2

3. Hardware port map

The full sensored Flex as of Section 6 of the curriculum. Sections 3 and 4 use only the day-one ports (motors and IMU); sensors get added in Section 6.

Port	Device	Notes
Smart 4	V5 Smart Motor — right drivetrain	Green (200 rpm) cartridge, 4" omni, direct drive. Forward polarity.
Smart 6	V5 Smart Motor — left drivetrain	Green cartridge, 4" omni, direct drive. Reversed (use {-6} in code). Verify on bench.
Smart 13	V5 Inertial Sensor (IMU)	Heading + tilt. EZ-Template needs this. Mount flat; calibration runs in initialize().
Smart 15	V5 Smart Motor — claw	Toggle pattern (open/close). MOTOR_BRAKE_HOLD + 1.8 A current limit prevents stall heat.
Smart 17	V5 Smart Motor — lift	Chain-driven. MOTOR_BRAKE_HOLD + 2.5 A current limit. Tare on boot via bottom limit (Section 6).
Smart 3	V5 Optical Sensor (Section 6)	Mounted in the claw throat. Proximity-gates smart grab; hue confirms block color.
Smart 7	V5 Distance Sensor (Section 6)	Mounted at back of robot. Used by D-pad ↓ park-against-wall macro.
ADI A	Bumper switch — lift bottom (Section 6)	Used to tare lift position on boot via lift_motor.tare_position().
ADI B	Bumper switch — lift top (Section 6)	Safety bound; prevents chain damage from over-extension.
ADI C	Potentiometer — lift pivot (Section 6)	Mounted on the lift pivot shaft, NOT the motor shaft. Enables closed-loop lift positions.

Polarity tip: 'Verify on bench' means push a positive voltage to each motor with the robot up on a block. If the lift moves down when commanded up, flip the polarity. If the chassis curves when driving forward, exactly one drive motor needs its polarity flipped. This is faster than guessing.

4. Controller mapping (final state)

This is the controller mapping after Section 7 (Driver-Assist Macros) is complete. Earlier sections use a subset — only sticks and R1/R2 lift in Section 4, everything else as students layer in macros and sensors.

Button	Action	Source
L Stick (Y)	Left drivetrain (analog)	opcontrol_tank() in Section 3
R Stick (Y)	Right drivetrain (analog)	opcontrol_tank() in Section 3
R1 (held)	Lift UP (soft-limit aware)	Section 4 base, Section 7 soft-limit
R2 (held)	Lift DOWN (soft-limit aware)	Section 4 base, Section 7 soft-limit
L1 (press)	Smart grab (auto-close if optical confirms)	Section 7 Macro 2
L2 (held)	Claw open — manual override	Section 4 base
A (press)	Pickup preset (lift GROUND + claw OPEN)	Section 7 Macro 1
Y (press)	Score preset (lift SCORE height)	Section 7 Macro 1
X (press)	Travel preset (lift CARRY + claw CLOSED)	Section 7 Macro 1
B (press)	Sequential score combo (drive → lift → release → back up)	Section 7 Macro 7
D-Pad ←	Precise 90° left turn (IMU)	Section 7 Macro 4
D-Pad →	Precise 90° right turn (IMU)	Section 7 Macro 4
D-Pad ↑	Slow-mode toggle (50% drivetrain)	Section 7 Macro 5
D-Pad ↓	Park-against-wall (distance sensor)	Section 6 Sensor 5

Failsafe: any joystick deflection > 30% during a running macro cancels it. Drivers are never trapped in a misbehaving routine. This pattern is documented in Section 7 of the curriculum and shown explicitly in the score_combo() helper.

5. Section walkthroughs (with key code)

Sections 1 and 2 are framing — the walkthroughs start with Section 3 (the first section students write code in). Each walkthrough shows the key code blocks and what students should walk away with after completing the section.

Section 3 — EZ-Template Flex Configuration

The first program students write. They set up the chassis constructor, the mechanism motors, and run a built-in drive_test auton to confirm the configuration works.

Key code — subsystems.hpp

■ include/subsystems.hpp

```
#pragma once
#include "EZ-Template/api.hpp"
#include "api.h"

// Two 11W motors, green (200 rpm) cartridges
// 4" omni wheels, direct drive (1:1)
inline ez::Drive chassis(
    {-6},       // Left motor: port 6, reversed
    {4},        // Right motor: port 4, forward
    13,         // IMU port
    4.0,        // Wheel diameter (inches)
    200,        // Green cartridge = 200 RPM
    1.0         // 1:1 external gear ratio
);

inline pros::Motor lift_motor(17);
inline pros::Motor claw_motor(15);
```

What students should walk away with

- **Port polarity is a code decision, not a hardware one.** The {-6} for left motor doesn't mean the cable is wired wrong — it means the motor is mounted with its output shaft facing the opposite direction from the right motor. The negative tells EZ-Template to flip the sign for that motor.
- **The five numbers in the Drive constructor matter in this order:** wheel diameter, cartridge RPM, gear ratio. Get them right once; every PID call inherits the calibration.
- **Always verify before tuning.** Run a built-in drive_test before changing any PID constants. If it curves, the polarity is wrong — fix that first.

Section 4 — Lift + Claw Control

First mechanism control on Flex. Two patterns — hold-to-move for the lift (motor runs while button held, brakes when released) and toggle for the claw (each press flips the state). Both use current limits to prevent stall damage.

Key code — lift (hold-to-move)

■ src/main.cpp — opcontrol() loop

```
if (master.get_digital(DIGITAL_R1)) {
    lift_motor.move(110);          // up at ~85% power
} else if (master.get_digital(DIGITAL_R2)) {
    lift_motor.move(-110);         // down at ~85% power
} else {
    lift_motor.brake();            // HOLD mode resists motion when released
}
```

Key code — claw (toggle)

■ src/main.cpp — file scope + opcontrol() loop

```
bool claw_closed = false;   // file scope

// inside opcontrol() while loop:
if (master.get_digital_new_press(DIGITAL_L1)) {
    claw_closed = !claw_closed;
}
claw_motor.move(claw_closed ? 90 : -90);
```

Key code — stall protection

■ src/main.cpp — initialize()

```
lift_motor.set_brake_mode(MOTOR_BRAKE_HOLD);
lift_motor.set_current_limit(2500);      // 2.5 A
claw_motor.set_brake_mode(MOTOR_BRAKE_HOLD);
claw_motor.set_current_limit(1800);      // 1.8 A — gentle grip
```

What students should walk away with

- **get_digital vs. get_digital_new_press is the bug-of-the-day.** The first returns true every loop iteration the button is held (use for hold-to-move). The second returns true once per press (use for toggles). Picking the wrong one is the most common student mistake in this section.
- **brake() with HOLD mode is what makes the lift hold position.** Without HOLD set in initialize(), brake() is just setting voltage to 0 — the lift falls under its own weight. The brake mode is configured once; the brake() call uses it.
- **Time-based lift is the floor, not the goal.** The 1.2-second timed lift only works on a fully-charged battery. Section 6's potentiometer is what makes it reliable.

Section 5 — Progressive Training Exercises

Six exercises ordered so each failure has exactly one possible cause — the thing students just changed. Run them in order. Skipping ahead breaks the diagnostic structure.

The six exercises

1	Drive forward 12 in	Confirm chassis polarity and basic PID work. ±2 in is acceptable first run.
2	Turn 90° right and back	Confirm IMU + turn PID. Front of robot should return to starting heading.
3	Drive a 24" × 24" square	Expose accumulated PID error. Well-tuned: returns within 2". Poorly-tuned: spirals out 6"+.
4	Timed lift to scoring height	Get comfortable with time-based motion. Identify battery-state failure modes — sets up Section 6.
5	Drive-lift-score combo	Multi-step auton. Record run-to-run variation; the variation IS the data.
6	Two-block scoring routine	Full auton complexity. Battery variation in lift will be visible; sets up sensor motivation.

What students should walk away with

- **Failure is data, not a setback.** The structured 'what happened / why / what sensor would fix it / what's the smallest test' pattern is what the engineering notebook scores. Push students to write all four after every failed exercise.
- **Run-to-run variation matters.** Exercise 5 specifically asks for three runs. The spread between them is what justifies sensor investment in Section 6.

Section 6 — Sensors to Add to Your Flex

Five sensors, added in a specific order: limit switches first (cheap, safety), potentiometer next (closed-loop lift), optical (auto-grab), distance (parking). Each sensor enables a specific capability that was guesswork before.

Key code — bottom limit zero-on-boot

■ src/main.cpp — initialize()

```
// Run the lift down until it hits the bottom switch, then tare position
while (!lift_bottom.get_value()) {
    lift_motor.move(-40);
    pros::delay(20);
}
lift_motor.brake();
lift_motor.tare_position();   // bottom = position 0 from here on
```

Key code — potentiometer-based lift position

■ src/main.cpp — file scope

```
constexpr int LIFT_GROUND      = 200;   // measured on bench
constexpr int LIFT_CARRY       = 900;
constexpr int LIFT_SCORE       = 1850;
constexpr int LIFT_TOLERANCE   = 30;

void lift_to_position(int target) {
    int err = target - lift_pot.get_value();
    while (std::abs(err) > LIFT_TOLERANCE) {
        int voltage = std::clamp(err / 8, -110, 110);
        lift_motor.move(voltage);
        pros::delay(20);
        err = target - lift_pot.get_value();
    }
    lift_motor.brake();
}
```

What students should walk away with

- **get_value() polarity gotcha on limit switches.** ADIDigitalIn returns 0 when pressed by default. Code that checks *get_value()* for 'is pressed' actually checks 'is released'. If the calibration loop never terminates, this is almost always the cause.
- **Pot constants are bench-measured, not guessed.** Students hand-move the lift to each named position and read the value from the brain screen. Update the code *and* the engineering notebook with the actual values.
- **Pot mounts on the pivot shaft, not the motor shaft.** If there's a gear ratio between the motor and the lift pivot (there is, on Flex), reading motor rotation gives wrong numbers. Mount the pot directly on the rotating element you want to measure.

Section 7 — Driver-Assist Macros

Seven macros that combine sensors with mechanism actions. The bridge from manual control to full auton. Driver presses one button; sensors decide the right thing to happen. This is the section where Flex starts feeling like a competition robot.

Key code — position presets (Macro 1)

■ src/main.cpp — file scope

```
void pickup_preset() {
    lift_to_position(LIFT_GROUND);
    claw_motor.move(-90);          // open
    pros::delay(400);
    claw_motor.brake();
    claw_closed = false;
}

void score_preset() {
    lift_to_position(LIFT_SCORE);  // claw state preserved
}

void travel_preset() {
    lift_to_position(LIFT_CARRY);
    claw_motor.move(90);           // close
    pros::delay(400);
    claw_motor.brake();
    claw_closed = true;
}
```

Key code — smart grab with failsafe (Macro 2)

■ src/main.cpp — file scope

```
void smart_grab() {
    if (claw_optical.get_proximity() > BLOCK_PROXIMITY_THRESHOLD) {
        claw_motor.move(90);
        pros::delay(500);
        claw_motor.brake();
        claw_closed = true;
        master.rumble(".");
    } else {
        pros::lcd::print(6, "Nothing to grab");
    }
}
```

Key code — sequential score combo (Macro 7)

■ src/main.cpp — file scope, with stick-deflection failsafe

```
void score_combo() {
    chassis.drive_set(60, 60);
    int elapsed = 0;
    while (rear_distance.get() > 300 && elapsed < 2000) {
        if (std::abs(master.get_analog(ANALOG_LEFT_Y)) > 30) {
            chassis.drive_set(0, 0);
            return;                                  // FAILSAFE
        }
        pros::delay(20);
        elapsed += 20;
    }
    chassis.drive_set(0, 0);
    lift_to_position(LIFT_SCORE);
    claw_motor.move(-90);
```

```
    pros::delay(400);
    claw_motor.brake();
    claw_closed = false;
    chassis.drive_set(-60, -60);
    pros::delay(600);
    chassis.drive_set(0, 0);
    master.rumble("-.");
}
```

What students should walk away with

- **Macros reduce cognitive load, not work.** The driver thinks 'score mode' instead of 'press lift-up, then release claw, then back up.' Same actions; one button. Cycle time drops from ~6s manual to ~4s macro. Two extra scoring cycles per match.
- **Every macro needs a failsafe.** A misbehaving macro that can't be cancelled is worse than no macro. The stick-deflection check in score_combo() is the pattern; copy it for any new macro the team writes.
- **opcontrol() ordering matters.** Drivetrain first (always running), one-shot macros next (new-press triggered), D-pad macros third, manual overrides last. Different order, different behavior.

6. Mentor implementation notes — where students get stuck

Six named traps from coaching Flex curricula. Each one has a specific symptom, a specific cause, and a specific fix. When you see the symptom, name the trap — students learn faster when they have a vocabulary for what went wrong.

Trap 1: The IMU calibration trap

Symptom: robot turns 70° when commanded to turn 90°. Or it veers off-heading during what should be straight driving.

Cause: the robot was touched (loaded onto field, repositioned) during the 2-second IMU calibration window in initialize(). The IMU recorded that motion as part of its zero-heading reference. Every heading reading is now off by the same amount.

Fix: plug in the brain, then walk away. Wait for the EZ-Template logo to show on the brain screen (that's the signal calibration is done) before touching the robot. Re-power if you suspect it happened mid-match.

Trap 2: Rising-edge confusion (get_digital vs. get_digital_new_press)

Symptom: the claw flutters open-close-open-close as the student holds L1. Or the slow-mode toggle flips 50 times in the time the student presses the D-pad once.

Cause: using *get_digital* for a toggle. It returns true every loop iteration the button is held — ~50 times per second.

Fix: *get_digital_new_press* for any toggle or one-shot action. *get_digital* only for hold-to-move (lift up/down, claw manual open).

Trap 3: Lift sag (brake mode not set)

Symptom: lift drops 4 inches every time the student releases R1. Section 4 code looks correct — brake() is called in the else branch.

Cause: motor brake mode defaults to COAST. *brake()* just sets voltage to 0 in COAST mode — the lift falls under gravity.

Fix: add *lift_motor.set_brake_mode(MOTOR_BRAKE_HOLD)* in initialize(). One line; permanent fix. Same for claw.

Trap 4: Chain skip on the lift (no soft limits)

Symptom: lift sometimes 'clicks' loudly when reaching the top of its travel. Eventually the chain skips or a sprocket tooth breaks. Often happens during the first competition, not in practice.

Cause: students added the top limit switch (Section 6, Sensor 2) but didn't add Section 7's soft-limit macro. The lift slams into the hard stop at full power, transmitting impact through the chain.

Fix: add the get_safe_lift_power() helper from Section 7 Macro 3. It decelerates the lift proactively as it approaches the limits. Driver doesn't notice it most of the time — that's the point.

Trap 5: Optical sensor 'always sees something'

Symptom: smart grab fires every time L1 is pressed, even when the claw is empty. Optical proximity reading is always above threshold.

Cause: the optical sensor is too close to the claw's own structure (it's seeing the claw itself), or its LED is pointed at a fixed surface that reflects back. Or BLOCK_PROXIMITY_THRESHOLD is set too low.

Fix: read the proximity value with an empty claw and again with a block. Threshold goes between those two numbers, closer to the empty value plus 20% margin. Document both numbers in the engineering notebook.

Trap 6: Polarity flip after a rebuild

Symptom: auton was working last week. After someone rebuilt the chassis, the robot now drives backward on every command. Or it curves where it used to go straight.

Cause: a drive motor was reinstalled with its gearbox facing the opposite direction. The {-6} convention assumes both motors face inward; if one was flipped, the polarity needs updating in code.

Fix: run the verify-on-bench check from Section 3. If both motors run backward, flip BOTH port signs. If only one runs backward, flip just that one.

7. Corrections log

This log accumulates corrections as mentors and students discover them. Each entry documents what changed, when, and why — useful for tracking how the curriculum evolves with use.

Initial release — May 15, 2026 — SW v309

Flex training curriculum created as a parallel to the existing clawbot-training. Six sections initially: Why Flex, Sensors On-Board, EZ-Setup, Lift + Claw, Progressive Exercises, Sensors to Add. Built against the v308 production baseline. Port map confirmed by Arturo: Left=6 (reversed), Right=4, IMU=13, Claw=15, Lift=17. Drive specs: two 11W green-cartridge motors, 4-inch omni wheels (team modification from stock mecanum), direct drive.

Expansion to 8 sections — May 15, 2026 — SW v310

Added Section 7 (Driver-Assist Macros) and Section 8 (From Flex to Custom Competition Robot) to match the expanded scope of clawbot-training. Macros section includes full PROS code for seven macros adapted to Flex's lift-and-claw geometry (score-combo replaces clawbot's pickup-combo as the sequential macro, since Override's repeated action is scoring blocks rather than picking them up). Section 8 inverts clawbot's 'Hero Bot for Override' framing — Flex IS the Hero Bot, so the natural forward-pointing section is the path to a custom competition robot.

To be filled in

Future corrections rounds will be logged here as mentors find them in use. Suggested format: section affected, symptom that triggered the correction, what was changed, and the SW version when the change shipped.

8. Quick reference card

Tear off this page and tape it inside the toolbox lid. The patterns students most often need to look up.

EZ-Template essentials

```
// Drive (in autonomous routines)
chassis.pid_drive_set(24_in, 110);   chassis.pid_wait();
chassis.pid_turn_set(90_deg, 110);   chassis.pid_wait();

// Driver control
chassis.opcontrol_tank();            // every loop iteration

// IMU and motor sensors
double heading = chassis.imu_get_rotation();
double lift_deg = lift_motor.get_position();
lift_motor.tare_position();          // zero the encoder
```

Mechanism control patterns

```
// Lift hold-to-move (with soft-limit helper from Section 7)
double v = get_safe_lift_power(110 if R1 else -110 if R2 else 0);
if (v != 0) lift_motor.move(v); else lift_motor.brake();

// Claw toggle (file scope: bool claw_closed)
if (master.get_digital_new_press(DIGITAL_L1)) claw_closed = !claw_closed;
claw_motor.move(claw_closed ? 90 : -90);

// Pot-based lift position
lift_to_position(LIFT_SCORE);        // closed-loop, defined in Section 6
```

Controller map (final state)

```
LEFT HAND                    RIGHT HAND
L1   smart grab              R1   lift up   (held, soft-limit)
L2   claw open  (held)       R2   lift down (held, soft-limit)
                             A    pickup preset
D← / D→  precise 90° turn     Y    score preset
D↑   slow-mode toggle        X    travel preset
D↓   park-against-wall       B    sequential score combo

L Stick: left drivetrain     R Stick: right drivetrain
Failsafe: stick > 30% during a macro cancels it.
```

Common pitfalls — fast triage

Symptom	Most likely cause
Turns 70° when commanded 90°	IMU was bumped during calibration. Re-power and don't touch.
Claw flutters when L1 held	Used get_digital instead of get_digital_new_press for toggle.
Lift sags 4" when released	Brake mode not set to HOLD in initialize().
Lift 'clicks' at top of travel	Missing soft-limit macro; lift slams hard stop.
Smart grab always fires	Threshold too low or optical sees its own structure.
Robot drives backward after rebuild	Drive motor reinstalled flipped; update port polarity.