TENNESSEE ROBOTICS CENTER · FTC PROGRAMMING GUIDE

Limelight 3A: Tracking Pollen and AprilTags

BIOBUZZ 2026–27 · Red 11161 · White 23866 · Blue 23867

Michael Puckett, Founder & Director · September 25, 2026

Overview

This guide takes a TRC FTC team from a Limelight 3A in the box to a robot that finds a yellow BIOBUZZ pollen ball, turns to face it, and drives to it.

It builds on the four-step Limelight flow from the BIOBUZZ Kickoff presentation (Mason Cox and Shaurya Nadagoudra): **Capture, Create a Pipeline, Get Data, Use It in Your Code.**

Who it's for: Red 11161, White 23866, and Blue 23867 programmers and anyone running the camera at practice. It assumes the team can already push code from Android Studio to the Control Hub.

What you'll have at the end:

- Saved Limelight pipelines for yellow, red and blue pollen, plus one for AprilTags
- A Java TeleOp that reads tx, ty, and ta and shows them on the Driver Station
- An auto-align and drive-to-pollen routine you can drop into TeleOp or autonomous, and code that switches between pipelines mid-match

1. Capture: hardware and setup

The camera has to be solid and have a clear view of the floor in front of the intake. Most tracking problems are really mounting problems.

1. **Mount it rigid.** Bolt the Limelight to the frame, not to a moving arm or turret. Any wobble shows up as jumpy tx values.
2. **Aim it down.** Tilt the camera down so pollen on the tiles fills the middle of the image from about 6 in. to 4 ft. in front of the robot. Write down the **mount height** (lens to floor, in inches) and the **mount angle** (degrees below horizontal). You need both in Step 3.
3. **Wire it.** Plug the Limelight 3A into a **USB 3.0 port on the Control Hub** with a short, strain-relieved cable. The camera draws its power from USB.
4. **Add it to the robot configuration.** On the Driver Station, open Configure Robot, scan, and name the device `limelight`. The name in the config must match the name in your code exactly.
5. **Open the web interface.** Plugged into a laptop over USB, go to **http://limelight.local:5801** (the address from the kickoff deck). On the robot, connect the laptop to the Control Hub's Wi-Fi and use the Control Hub address from Limelight's FTC docs.
6. **Update the firmware first.** Use the latest Limelight OS before you tune anything. A firmware update can reset pipelines, so do it now and not the night before a qualifier.

2. Create the yellow pollen pipeline

Use a **Color/Retroreflective** pipeline in slot 0. It finds every pixel inside a color range, then keeps only the blobs that are shaped like a ball. Put real pollen on real field tiles under your practice lights while you tune.

1. In the web interface, select **Pipeline 0**, rename it `pollen-yellow`, and set the type to **Color/Retroreflective**.
2. **Input tab:** lower **Exposure** until the image looks slightly dark and the balls look saturated. Low exposure cuts motion blur and glare. Raise **Black Level** a little if the tiles still show up.
3. **Thresholding tab:** click a pollen ball with the eyedropper/wand to seed the range, then adjust with the starting values below. Turn on the threshold view: pollen should be solid white and everything else black.
4. **Contour filtering tab:** set the filters below so only round, ball-sized blobs survive. Sort by **Largest**, so the closest ball is the target.
5. **Output tab:** leave targeting at the center of the target. Click **Save**, then download the pipeline file and commit it to the team GitHub so everyone has the same tuning.

Starting values (tune from here, every field is different):

Setting	Start at	Why
Hue	18–35	Yellow on Limelight's 0–180 hue scale
Saturation	110–255	Drops white field walls and gray tiles
Value	100–255	Drops dark shadows under the robot
Erode / Dilate	1 / 2	Removes speckle, fills holes from ball dimples
Area (% of image)	0.05–40	Ignores tiny specks and far-away balls
Fullness	50–100%	A ball fills most of its bounding box
Aspect ratio (W/H)	0.6–1.6	A ball is roughly square; a strip of tape is not
Sort	Largest	Closest ball wins

> **Watch the green balls.** Lime-green and yellow sit close together on the hue wheel, so if the green ball lights up in threshold view, **lower the top of the Hue range** a step at a time until only yellow stays white.

3. Get the data into Java

The Limelight 3A is built into the FTC SDK, so you don't need an extra library. Every loop, the camera hands you three numbers for the biggest yellow blob:

Value	Meaning	Use it to
tx	Degrees left (−) or right (+) of the crosshair	Turn to face the ball
ty	Degrees above (+) or below (−) the crosshair	Estimate distance
ta	Percent of the image the ball fills	Tell when the ball is close enough to intake

Start with this test TeleOp. It shows the numbers on the Driver Station so the team can check the pipeline before trusting it to drive.

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.hardware.limelightvision.LLResult;
import com.qualcomm.hardware.limelightvision.Limelight3A;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;

@TeleOp(name = "Pollen Vision Test", group = "Vision")
public class PollenVisionTest extends LinearOpMode {

    // Measure these on YOUR robot (Step 1)
    static final double CAMERA_HEIGHT_IN = 8.0;   // lens to floor
    static final double MOUNT_ANGLE_DEG  = 25.0;  // tilted down from horizontal
    static final double BALL_RADIUS_IN   = 2.5;   // measure a pollen ball

    @Override
    public void runOpMode() {
        Limelight3A limelight = hardwareMap.get(Limelight3A.class, "limelight");
        limelight.setPollRateHz(100);   // ask for fresh data 100x per second
        limelight.pipelineSwitch(0);    // pollen-yellow pipeline
        limelight.start();

        waitForStart();

        while (opModeIsActive()) {
            LLResult result = limelight.getLatestResult();

            if (result != null && result.isValid()) {
                double tx = result.getTx();
                double ty = result.getTy();
                double ta = result.getTa();

                telemetry.addData("Pollen", "FOUND");
                telemetry.addData("tx (deg)", "%.1f", tx);
                telemetry.addData("ty (deg)", "%.1f", ty);
                telemetry.addData("ta (%)", "%.2f", ta);
                telemetry.addData("distance (in)", "%.1f", distanceToBall(ty));
            } else {
                telemetry.addData("Pollen", "none in view");
            }
            telemetry.update();
        }
        limelight.stop();
    }

    // Camera looks down at MOUNT_ANGLE_DEG; the ball sits ty degrees above or below the crosshair
    double distanceToBall(double ty) {
        double angleDownRad = Math.toRadians(MOUNT_ANGLE_DEG - ty);
        return (CAMERA_HEIGHT_IN - BALL_RADIUS_IN) / Math.tan(angleDownRad);
    }
}
```

The numbers in the three constants are placeholders. Replace them with measurements from your robot and a real pollen ball, then check the distance readout against a tape measure at 1, 2, and 3 ft.

4. Use it: align and drive to pollen

The kickoff deck's rule is the whole idea: **tx = +8° means the target is to the right, so turn right until tx is about 0°.** A proportional (P) controller does that smoothly. Turn power is tx times a small gain, so the robot turns hard when it's far off and slows as it lines up.

Add these to the test OpMode and call `chasePollen()` while the driver holds a button (TeleOp assist) or inside an autonomous action.

```
// Tuning constants: start small, raise until it's quick without wobbling
static final double TURN_KP      = 0.02;  // power per degree of tx
static final double DRIVE_KP     = 0.03;  // power per inch of distance
static final double TX_TOLERANCE = 2.0;   // degrees: "lined up"
static final double STOP_DIST_IN = 8.0;   // stop here and let the intake grab it
static final double MAX_POWER    = 0.5;

/** Returns true when the robot is lined up and close enough to intake. */
boolean chasePollen(Limelight3A limelight) {
    LLResult result = limelight.getLatestResult();
    if (result == null || !result.isValid()) {
        drive(0, 0, 0.25);            // no ball: slow search spin
        return false;
    }

    double tx = result.getTx();
    double dist = distanceToBall(result.getTy());

    double turn = clip(tx * TURN_KP);
    // Only drive forward once we're mostly facing the ball
    double forward = Math.abs(tx) < 10 ? clip((dist - STOP_DIST_IN) * DRIVE_KP) : 0;

    drive(forward, 0, turn);          // forward, strafe, turn
    return Math.abs(tx) < TX_TOLERANCE && dist <= STOP_DIST_IN;
}

double clip(double p) { return Math.max(-MAX_POWER, Math.min(MAX_POWER, p)); }

// Connect to YOUR drivetrain: your mecanum drive method, or Pedro Pathing's TeleOp drive
void drive(double forward, double strafe, double turn) { /* team drive code */ }
```

Tuning order:

1. Set `DRIVE_KP` to 0 and tune turning only. Raise `TURN_KP` until the robot snaps to the ball. Back it off 20% if it wobbles back and forth.
2. Turn driving back on. Raise `DRIVE_KP` until it approaches quickly and stops at `STOP_DIST_IN` without bumping the ball away.
3. If the turn direction is backwards, flip the sign of `turn`. That's a drivetrain convention issue, not a camera issue.

In autonomous: use Pedro Pathing to drive to the area where pollen starts, then hand off to `chasePollen()` for the final few feet. Wrap it in a RoadRunner-style action that finishes when it returns true, so it chains into your Sequential and Parallel actions like the kickoff deck showed.

5. Red and blue pollen

Each pollen color gets its own pipeline. Build yellow first (Step 2), then copy it into two more slots so the shape filters carry over, and change only the color range.

Slot	Pipeline name	Hue to start with	Common false detections
0	`pollen-yellow`	18–35	Lime-green balls
1	`pollen-red`	0–8 (or 170–180)	Skin, red shirts, red walls and bumpers
2	`pollen-blue`	100–125	Blue shirts, jeans, blue field tape

Keep saturation high on all three (start at 110–255). Clothing and skin are much less saturated than a molded game ball, so this filter does most of the work.

> **Red is the hard one.** On the Limelight's 0–180 hue scale, red sits at both ends: near 0 and near 180. Start with 0–8 in threshold view. If parts of the ball drop out, try 170–180, or use the invert-hue option if your Limelight OS version has one. Tune red last and give it the most time.

Switching colors in code. Keep the slot numbers in one place and switch with `pipelineSwitch()`. After a switch, skip any result that still comes from the old pipeline:

```
static final int YELLOW = 0, RED = 1, BLUE = 2;

void track(Limelight3A limelight, int slot) {
    limelight.pipelineSwitch(slot);
}

LLResult freshResult(Limelight3A limelight, int slot) {
    LLResult r = limelight.getLatestResult();
    // Ignore the frame or two that still come from the previous pipeline
    if (r == null || !r.isValid() || r.getPipelineIndex() != slot) return null;
    return r;
}
```

In TeleOp, a driver button can pick the color. On most FTC gamepads the face buttons already match: **Y = yellow, B = red, X = blue.**

6. AprilTags and switching pipelines

The Limelight runs one pipeline at a time. It can't track pollen and read AprilTags in the same frame, but switching takes only a frame or two. So switch based on what the robot is doing:

Robot is…	Pipeline	What it gives you
Starting autonomous	AprilTag (slot 3)	Where the robot is on the field

Robot is…	Pipeline	What it gives you
Collecting	Pollen color (slot 0, 1 or 2)	Angle and distance to the nearest ball
Lining up to score	AprilTag (slot 3)	Angle to the goal's tag

Set up the AprilTag pipeline:

1. Select **Pipeline 3**, name it `apriltags`, and set the type to **AprilTag** (Fiducial).
2. Set the tag family to **36h11**, the family FTC uses, and enter the tag size from the BIOBUZZ game manual.
3. Keep the image bright and sharp. Tags need crisp black-and-white edges, so they want more exposure than the color pipelines.
4. For full field position (botpose), upload the season's field map and enter the camera's position on the robot. Skip this until basic tag reading works.

Read tags in Java. With the AprilTag pipeline active, tx and ty point at a tag, and each tag in view reports its ID:

```
import com.qualcomm.hardware.limelightvision.LLResultTypes;

static final int TAGS = 3;

limelight.pipelineSwitch(TAGS);
LLResult r = freshResult(limelight, TAGS);
if (r != null) {
    for (LLResultTypes.FiducialResult tag : r.getFiducialResults()) {
        telemetry.addData("Tag " + tag.getFiducialId(), "%.1f deg left/right",
                tag.getTargetXDegrees());
    }
}
```

Example autonomous flow: read a tag at the start to confirm position → switch to the pollen color → `chasePollen()` → switch back to AprilTags → aim at the goal and score.

> **Try it:** the MakerFest demo program uses all four pipelines. Y, B and X track yellow, red and blue pollen, and A reads AprilTags. It's a quick way for a new programmer to watch pipeline switching work before building it into a robot.

7. Test checklist and troubleshooting

Run this checklist at every new venue. Gym and ballroom lighting are very different from TRC's labs, so re-check the threshold during practice-field time.

- ☐ One pollen ball, centered, at 1 ft: telemetry says FOUND and tx is within 2° of 0
- ☐ Ball moved left, then right: tx goes negative, then positive
- ☐ Distance readout within 2 in. of a tape measure at 1, 2, and 3 ft
- ☐ Pile of mixed balls: only the selected color lights up in threshold view

- ☐ Robot driving at full speed: target doesn't drop out (if it does, lower exposure)
- ☐ Drive team's shirts, the field wall, and the alliance station aren't detected
- ☐ Pipeline files exported and committed to GitHub; venue tweaks noted in the engineering notebook

Problem	Likely cause	Fix
Nothing detected	Wrong pipeline, or the device name doesn't match	Check `pipelineSwitch()` and that the config name is `limelight`
Green balls detected too	Hue range too wide	Lower the top of the Hue range
Target flickers on and off	Exposure too high, or area/fullness filters too tight	Lower exposure; widen Area and Fullness a little
tx jumps around while stopped	Camera mount flexes	Stiffen the mount; move it off the turret or arm
Robot overshoots and wobbles	TURN_KP too high	Cut TURN_KP by 20–30%
Robot turns away from the ball	Turn sign reversed for your drivetrain	Flip the sign of `turn`
Works at TRC, fails at the event	Different lighting	Retune the Value and Saturation minimums during field time