Concept Samples Documentation

Table of Contents

1. AprilTag Concepts
2. Gamepad Concepts
3. Hardware & Motor Concepts
4. Sound Concepts
5. Vision & Color Concepts
6. Miscellaneous Concepts

AprilTag Concepts

These samples demonstrate how to use AprilTags for vision-based navigation and localization.

- [x] ~~ConceptAprilTagSwitchableCameras.java~~
- [x] ~~ConceptAprilTag.java~~
- [x] ~~ConceptAprilTagEasy.java~~
- [x] ~~ConceptAprilTagMultiPortal.java~~
- [x] ~~ConceptAprilTagLocalization.java~~
- [x] ~~ConceptAprilTagOptimizeExposure.java~~

ConceptAprilTag.java

Description:

This is the fundamental sample for AprilTag detection. It shows how to configure the `AprilTagProcessor` and `VisionPortal` manually using Builders.

Key Features:

- **Configuration:** Demonstrates setting up camera parameters (lens intrinsics), tag families, and output units.
- **Telemetry:** Displays detected tag IDs and their pose information (XYZ position, Pitch/Roll/Yaw orientation, Range/Bearing/Elevation).
- **CPU Optimization:** Shows how to stop/resume streaming to save CPU resources when vision is not needed.

Code Breakdown:

- **Initialization (`initAprilTag`):**

- `new AprilTagProcessor.Builder()...build()` : Creates the processor. You can uncomment lines to customize the tag family (e.g., `TAG_36h11`), output units (Inches/Degrees), or lens intrinsics.

- `new VisionPortal.Builder()...build()` : Creates the portal. It sets the camera (`Webcam 1` or `BuiltinCameraDirection.BACK`), resolution, and adds the `aprilTag` processor.

- **Main Loop (`runOpMode`):**

- `telemetryAprilTag()` : Called repeatedly to get detections and print them.

- `visionPortal.stopStreaming()` / `resumeStreaming()` : Controlled by D-pad Up/Down to manually toggle the camera stream, saving CPU when not needed.

- **Telemetry (`telemetryAprilTag`):**

- `aprilTag.getDetections()` : Returns a list of `AprilTagDetection` objects.

- Iterates through detections:

- If `metadata` is not null (tag is in the library), it prints `ftcPose` data (XYZ, PRY, RBE).

- If `metadata` is null, it prints raw center pixel coordinates.

ConceptAprilTagEasy.java

Description:

A simplified version of `ConceptAprilTag` that uses default settings for quick setup.

Key Features:

- **Easy Initialization:** Uses `AprilTagProcessor.easyCreateWithDefaults()` and `VisionPortal.easyCreateWithDefaults()` to minimize boilerplate code.
- **Functionality:** Same detection and telemetry capabilities as the standard concept but with less configuration flexibility.

Code Breakdown:

- **Initialization (`initAprilTag`):**

- `AprilTagProcessor.easyCreateWithDefaults()` : Creates a processor with standard settings (Tag36h11, default units).

- `VisionPortal.easyCreateWithDefaults(...)` : Creates the portal and automatically attaches the camera and processor.

- **Main Loop:** Identical to `ConceptAprilTag`, calling `telemetryAprilTag()` and handling stream toggling.

ConceptAprilTagMultiPortal.java

Description:

Demonstrates how to use two separate cameras simultaneously by creating two `VisionPortal` instances.

Key Features:

- **Multi-Camera:** Shows how to split the screen layout using `VisionPortal.makeMultiPortalView()` to display two camera feeds.
- **Parallel Processing:** Runs independent AprilTag processors on each camera.

Code Breakdown:

- **View Splitting:**

- `VisionPortal.makeMultiPortalView(2, ...)`: Requests two view IDs for the camera monitor.

- `portal1ViewId` / `portal2ViewId`: These IDs are passed to the VisionPortal builders.

- **Dual Processors:**

- Creates `aprilTagProcessor1` and `aprilTagProcessor2` independently.

- **Dual Portals:**

- Builds `portal1` for "Webcam 1" using `portal1ViewId`.

- Builds `portal2` for "Webcam 2" using `portal2ViewId`.

- **Telemetry:**

- Displays the count of tags detected by each processor separately (`aprilTagProcessor1.getDetections().size()`).

ConceptAprilTagLocalization.java

Description:

Focuses on using AprilTags to determine the robot's absolute position on the field.

Key Features:

- **Camera Offset:** Explains how to define the camera's position and orientation relative to the robot's center using `Position` and `YawPitchRollAngles`.

- **Field Pose:** Calculates and displays the robot's field-relative coordinates (X, Y, Z, Heading) based on the detected tag's known location.

Code Breakdown:

- **Camera Pose Definition:**

- `cameraPosition`: Defines the XYZ offset of the camera lens from the robot's center (e.g., 6 inches forward).

- `cameraOrientation`: Defines the rotation of the camera (e.g., `pitch = -90` for a camera facing forward but mounted horizontally).

- `.setCameraPose(cameraPosition, cameraOrientation)`: Passed to the `AprilTagProcessor.Builder`.

- **Telemetry:**

- Accesses `detection.robotPose`: This object contains the transformed field coordinates (`getPosition()`) and orientation (`getOrientation()`) of the *robot*, not just the tag relative to the camera.

ConceptAprilTagOptimizeExposure.java

Description:

A utility OpMode to help tune camera exposure and gain settings for optimal AprilTag detection, especially to reduce motion blur.

Key Features:

- **Manual Control:** Allows using the gamepad to adjust exposure (ms) and gain on the fly.
- **Optimization:** Helps find the lowest exposure time that still yields reliable detection, which is crucial for detecting tags while moving.

Code Breakdown:

- **Camera Controls:**

- `visionPortal.getCameraControl(ExposureControl.class)`: Accesses exposure settings.

- `visionPortal.getCameraControl(GainControl.class)`: Accesses gain settings.

- **Manual Adjustment Loop:**

- Reads gamepad bumpers/triggers to increment/decrement `myExposure` and `myGain`.

- `setManualExposure(myExposure, myGain)`: Applies the new values.

- Checks `exposureControl.getMode()` and switches to `Mode.Manual` if needed.

- **Telemetry:** Displays the current Exposure (ms) and Gain values alongside the number of detected tags.

ConceptAprilTagSwitchableCameras.java

Description:

Shows how to switch between two cameras connected to the same `VisionPortal` during runtime.

Key Features:

- **Switchable Camera:** Uses `ClassFactory.getInstance().getCameraManager().nameForSwitchableCamera()` to combine two webcams into a single logical camera.
- **Runtime Switching:** Allows the driver to toggle the active camera using gamepad buttons.

Code Breakdown:

- **Initialization:**

- `ClassFactory...nameForSwitchableCamera(webcam1, webcam2)` : Creates a virtual "Switchable Camera" name.

- The `VisionPortal` is built using this switchable camera name.

- **Switching Logic (`doCameraSwitching`):**

- Monitors gamepad bumpers.

- `visionPortal.setActiveCamera(webcam1)` : Switches the active stream to Webcam 1.

- `visionPortal.setActiveCamera(webcam2)` : Switches to Webcam 2.

- **Telemetry:** Indicates which camera is currently active.

Gamepad Concepts

These samples explore advanced features of the gamepad controllers.

- [x] ~~ConceptGamepadEdgeDetection.java~~
- [x] ~~ConceptGamepadRumble.java~~
- [x] ~~ConceptGamepadTouchpad.java~~

ConceptGamepadEdgeDetection.java

Description:

Demonstrates how to detect "edges" of button presses (rising edge = pressed, falling edge = released) rather than just the current state.

Key Features:

- **Edge Detection:** Uses methods like `gamepad1.leftBumperWasPressed()` to trigger an action only once per press, preventing repeated triggers while holding a button.

Code Breakdown:

- **Main Loop:**

- `gamepad1.leftBumperWasPressed()` : Returns `true` ONLY on the frame the button transitions from unpressed to pressed.

- `gamepad1.leftBumperWasReleased()` : Returns `true` ONLY on the frame the button transitions from pressed to unpressed.

- `gamepad1.left_bumper` : Returns the standard current state (true if held down).

- **Telemetry:** Compares the edge detection methods with the standard state check to show the difference.

ConceptGamepadRumble.java

Description:

Illustrates how to use the rumble (vibration) feature on supported gamepads (Xbox, PS4).

Key Features:

- **Rumble Effects:** Shows how to trigger simple rumbles, timed rumbles, and custom rumble patterns (e.g., "blips" or complex sequences).
- **Feedback:** Useful for providing tactile feedback to the driver (e.g., "Endgame started", "Object collected").

Code Breakdown:

- **Simple Rumble:**

- `gamepad1.rumble(0.9, 0, 200)` : Rumbles left motor at 90% power for 200ms.

- `gamepad1.rumbleBlips(3)` : Triggers a built-in effect of 3 short pulses.

- **Custom Effects:**

- `new Gamepad.RumbleEffect.Builder()...build()`: Creates a sequence of rumble steps.
- `.addStep(1.0, 1.0, 500)`: Adds a step (Left 100%, Right 100%, 500ms).
- `gamepad1.runRumbleEffect(effect)`: Plays the custom sequence.

- **Management:**
- `gamepad1.isRumbling()`: Checks if a rumble is currently active.
- `gamepad1.stopRumble()`: Immediately cancels any active rumble.

ConceptGamepadTouchpad.java

Description:

Shows how to read input from the touchpad found on some controllers (like the Sony PS4 DualShock).

Key Features:

- **Touch Coordinates:** Reads X and Y coordinates for up to two fingers.
- **Gestures:** Can be used to implement swipe gestures or precise analog inputs.

Code Breakdown:

- **Finger Detection:**
- `gamepad1.touchpad_finger_1`: Boolean, true if the first finger is touching.
- `gamepad1.touchpad_finger_2`: Boolean, true if a second finger is touching.

- **Coordinates:**
- `gamepad1.touchpad_finger_1_x` / `_y`: Returns float values (-1.0 to 1.0) representing the position on the touchpad.

- **Telemetry:** Displays the status and coordinates of detected touches.

Hardware & Motor Concepts

These samples cover motor control, servo operation, and other hardware interactions.

- [x] ~~ConceptMotorBulkRead.java~~
- [x] ~~ConceptRampMotorSpeed.java~~
- [x] ~~ConceptRevLED.java~~

- [x] ~~ConceptRevSPARKMini.java~~
- [x] ~~ConceptScanServo.java~~
- [x] ~~ConceptLEDStick.java~~

ConceptMotorBulkRead.java

Description:

CRITICAL for performance. Demonstrates how to use "Bulk Reads" to significantly speed up loop times by reducing the number of hardware transactions.

Key Features:

- **Caching Modes:** Compares `OFF` (slowest), `AUTO` (automatic caching), and `MANUAL` (fastest but requires manual clearing) modes.
- **Optimization:** Explains why reading multiple encoders individually is slow and how bulk reads fetch all data in one go.

Code Breakdown:

- **LynxModule Access:**

- `hardwareMap.getAll(LynxModule.class)` : Retrieves all Expansion/Control Hubs.

- **Mode Selection:**

- `module.setBulkCachingMode(LynxModule.BulkCachingMode.AUTO)` : Clears cache automatically when a write occurs or a new read is requested.

- `module.setBulkCachingMode(LynxModule.BulkCachingMode.MANUAL)` : The most efficient mode. You MUST call `clearBulkCache()` once per loop.

- **Performance Test:**

- The code runs a loop reading 4 encoders.

- It measures the time taken for the loop in different modes (OFF, AUTO, MANUAL) to demonstrate the speed difference (often 2-3x faster with MANUAL).

ConceptRampMotorSpeed.java

Description:

A simple example of ramping motor power up and down.

Key Features:

- **Linear Ramping:** Gradually increases and decreases motor power to avoid sudden jerks or voltage spikes.

Code Breakdown:

- **Ramping Logic:**

- `power += RAMP_INCREMENT` : Increases power by a small step in each loop iteration.
- `power -= RAMP_INCREMENT` : Decreases power.
- `Range.clip(power, -1.0, 1.0)` : Ensures power stays within valid limits.

- **Cycle Time:** Uses `sleep(CYCLE_MS)` to control the speed of the ramp.

ConceptRevLED.java

Description:

Shows how to control the REV Digital Indicator (Red/Green LED module).

Key Features:

- **Digital Channels:** Configures the LED module as two digital channels to control Red and Green independently (or mix for Amber).

Code Breakdown:

- **Hardware Mapping:**

- `hardwareMap.get(LED.class, "green")` and `hardwareMap.get(LED.class, "red")` : Accesses the two channels of the device.

- **Control:**

- `green.on()` / `green.off()` : Turns the specific color channel on or off.
- `red.on()` / `red.off()` : Turns the red channel on or off.
- Turning both on results in an amber/orange color.

ConceptRevSPARKMini.java

Description:

Demonstrates how to use the REV SPARKmini motor controller.

Key Features:

- **Servo Port Control:** The SPARKmini is controlled via a Servo port but drives a DC motor.
- **Simple Drive:** Shows a basic tank/POV drive implementation using these controllers.

Code Breakdown:

- **Initialization:**

  - `hardwareMap.get(DcMotorSimple.class, "left_drive")` : The SPARKmini is treated as a `DcMotorSimple` in the configuration, even though it plugs into a servo port.

- **Drive Control:**

  - Calculates `leftPower` and `rightPower` based on gamepad inputs (POV Mode).
  - `leftDrive.setPower(leftPower)` : Sets the speed (-1.0 to 1.0).

ConceptScanServo.java

Description:

A basic utility to sweep a servo back and forth.

Key Features:

- **Testing:** Useful for testing servo range of motion and mechanical linkages.

Code Breakdown:

- **Scanning Logic:**

  - `position += INCREMENT` : Moves the servo position by a small amount each cycle.
  - Checks if `position >= MAX_POS` or `position <= MIN_POS` to reverse direction.
  - `servo.setPosition(position)` : Updates the physical servo position.

ConceptLEDStick.java

Description:

Controls the SparkFun QWIIC LED Stick (a strip of addressable LEDs).

Key Features:

- **Color Control:** Sets colors for individual LEDs or the entire strip.
- **Brightness:** Adjusts brightness levels.
- **Driver Feedback:** Can be used to signal robot status (e.g., "Ready to fire").

Code Breakdown:

- **Initialization:**

  - `hardwareMap.get(SparkFunLEDStick.class, "led_stick")` : Accesses the device.
  - `ledStick.setBrightness(20)` : Sets global brightness (0-31).

- **Color Setting:**

- `ledStick.setColor(index, Color.GREEN)` : Sets a specific pixel (0-9) to a color.
- `ledStick.setColor(Color.RED)` : Sets ALL pixels to red.
- `ledStick.turnAllOff()` : Turns off all LEDs.

Sound Concepts

These samples show how to play audio files on the Robot Controller or Driver Station.

- ~~ConceptSoundsASJava.java~~
- ~~ConceptSoundsOnBotJava.java~~
- ~~ConceptSoundsSKYSTONE.java~~

ConceptSoundsASJava.java

Description:

For Android Studio users. Plays sounds stored as raw resources in the app.

Key Features:

- **Resource Loading:** Uses `R.raw.filename` to access sound files compiled into the APK.
- **SoundPlayer:** Uses `SoundPlayer` to play these resources.

Code Breakdown:

- **Resource ID Lookup:**

- `hardwareMap.appContext.getResources().getIdentifier("silver", "raw", ...)` : Dynamically gets the integer ID for the resource file `silver.wav` located in `res/raw`.

- **Preloading:**

- `SoundPlayer.getInstance().preload(...)` : Loads the sound into memory before the match starts to prevent lag during playback.

- **Playback:**

- `SoundPlayer.getInstance().startPlaying(...)` : Triggers the sound.
- Checks `silverFound` flag to ensure the resource exists before trying to play it.

ConceptSoundsOnBotJava.java

Description:

For OnBotJava users. Plays sounds stored as files on the Robot Controller's storage.

Key Features:

- **File Access:** Loads `.wav` files from a specific directory (e.g., `/FIRST/blocks/sounds`).
- **Flexibility:** Allows adding new sounds without recompiling the app.

Code Breakdown:

- **File Definition:**

- `new File("/sdcard" + soundPath + "/gold.wav")`: Creates a Java `File` object pointing to the sound file on the internal storage.

- `goldFile.exists()`: Verifies the file is actually there.

- **Playback:**

- `SoundPlayer.getInstance().startPlaying(..., goldFile)`: Plays the sound directly from the file object.

ConceptSoundsSKYSTONE.java

Description:

Demonstrates playing built-in sound resources provided by the SDK (Star Wars themed from Skystone season).

Key Features:

- **Built-in Library:** Accesses a library of pre-loaded sounds.
- **Async Playback:** Shows how to play sounds without blocking the main loop.

Code Breakdown:

- **Sound List:**

- `String sounds[] = {"ss_alarm", ...}`: A list of filenames available in the SDK resources.

- **Selection Logic:**

- Uses D-pad Up/Down to cycle through the `sounds` array index.

- **Async Playback with Callback:**

- `SoundPlayer.getInstance().startPlaying(..., new Runnable() { ... })`: Plays the sound and runs the code in `run()` when it finishes.

- This is used to clear the `soundPlaying` flag, preventing the user from triggering another sound while one is already playing.

Vision & Color Concepts

These samples demonstrate using the camera for color detection and analysis.

- ~~ConceptVisionColorLocator_Circle.java~~
- ~~ConceptVisionColorLocator_Rectangle.java~~
- ~~ConceptVisionColorSensor.java~~

ConceptVisionColorLocator_Circle.java

Description:

Uses OpenCV to find circular blobs of a specific color.

Key Features:

- **ColorBlobLocatorProcessor:** Configures the processor to find blobs matching a color range.
- **Filtering:** Filters results by criteria like area, circularity, and density to find the correct object.
- **Visualization:** Draws contours and fitting circles on the camera preview.

Code Breakdown:

- **Processor Setup:**
  - `new ColorBlobLocatorProcessor.Builder()` : Starts the builder.
  - `.setTargetColorRange(ColorRange.BLUE)` : Sets the color to look for.
  - `.setContourMode(EXTERNAL_ONLY)` : Ignores holes inside blobs.
  - `.setBlurSize(5)` : Blurs the image slightly to reduce noise.
  - `.setDilateSize(15)` / `.setErodeSize(15)` : Uses morphological operations to close gaps in the blobs.
- **Filtering Loop:**
  - `colorLocator.getBlobs()` : Gets all detected blobs.
  - `ColorBlobLocatorProcessor.Util.filterByCriteria(...)` : Removes blobs that don't match criteria.

- `BY_CONTOUR_AREA`: Removes tiny noise or huge artifacts.
- `BY_CIRCULARITY`: Keeps only round-ish objects (1.0 is perfect circle).

- **Data Access:**

- `blob.getCircle()`: Returns the `Circle` object fitting the blob.
- `circle.center` / `circle.radius`: The position and size of the object.

ConceptVisionColorLocator_Rectangle.java

Description:

Similar to the Circle locator but optimized for finding rectangular colored regions.

Key Features:

- **Box Fit:** Fits a rotated rectangle to the detected blob.
- **Aspect Ratio:** Can filter blobs based on their aspect ratio (e.g., to distinguish a game element from a random patch of color).

Code Breakdown:

- **Processor Setup:**

- Similar to the Circle version but often uses `setBoxFitColor` for visualization.

- **Filtering:**

- `BY_ASPECT_RATIO`: Useful for distinguishing squares (ratio ~1) from long rectangles.

- **Data Access:**

- `blob.getBoxFit()`: Returns a `RotatedRect`.
- `boxFit.center`: The center point.
- `boxFit.angle`: The rotation angle of the rectangle.
- `boxFit.size`: Width and height.

ConceptVisionColorSensor.java

Description:

Uses the camera as a "virtual" color sensor to determine the predominant color in a specific region.

Key Features:

- **PredominantColorProcessor:** Analyzes a Region of Interest (ROI) to find the dominant color.
- **Swatches:** Matches the detected color to a list of predefined "swatches" (e.g., Red, Blue, Yellow) for easy classification.

Code Breakdown:

- **Processor Setup:**

- `new PredominantColorProcessor.Builder()`: Starts the builder.

- `.setRoi(...)`: Defines the sub-region of the image to analyze (e.g., the center 10%).

- `.setSwatches(...)`: Defines the list of valid colors to report (e.g., `RED`, `BLUE`, `YELLOW`).

- **Analysis:**

- `colorSensor.getAnalysis()`: Returns a `Result` object.

- `result.closestSwatch`: The enum value of the matched color.

- `result.RGB`: The raw average RGB values of the region.

Miscellaneous Concepts

Other useful concepts and utilities.

- [x] ~~ConceptBlackboard.java~~
- [x] ~~ConceptExploringIMUOrientation.java~~
- [x] ~~ConceptNullOp.java~~
- [x] ~~ConceptTelemetry.java~~

ConceptBlackboard.java

Description:

Demonstrates how to pass data between different OpModes (e.g., from Autonomous to TeleOp).

Key Features:

- **Persistence:** Stores key-value pairs in a "Blackboard" that survives OpMode transitions (but not app restarts).
- **Usage:** Useful for passing the final robot pose or alliance color from Auto to TeleOp.

Code Breakdown:

- **Accessing Blackboard:**

- `blackboard`: A built-in field in `OpMode` (like `telemetry` or `hardwareMap`).

- **Reading Data:**

- `blackboard.getOrDefault("key", defaultVal)`: Safely retrieves a value or a default if missing.

- `blackboard.get("key")`: Retrieves the value (returns null if missing).

- **Writing Data:**

- `blackboard.put("key", value)`: Stores a value.

- The sample increments a "Times started" counter in `init()` and lets you set an "Alliance" value with bumpers in `loop()`.

ConceptExploringIMUOrientation.java

Description:

A utility tool to help determine the correct Logo/USB orientation settings for the Control Hub's IMU.

Key Features:

- **Interactive Setup:** Allows changing orientation parameters via gamepad and observing the resulting Pitch/Roll/Yaw values to verify correctness.

Code Breakdown:

- **IMU Initialization:**

- `RevHubOrientationOnRobot`: A helper class to define how the Hub is mounted.

- `new RevHubOrientationOnRobot(logoDirection, usbDirection)`: Creates the orientation object.

- `imu.initialize(...)`: Re-initializes the IMU with the new parameters.

- **Interactive Loop:**

- Uses gamepad bumpers to cycle through `LogoFacingDirection` values (UP, DOWN, LEFT, RIGHT, etc.).

- Uses triggers to cycle through `UsbFacingDirection` values.

- Calls `updateOrientation()` whenever a change is detected.

- **Verification:**

- Displays Pitch, Roll, and Yaw. The user can physically tilt the robot to see if the values change as expected (e.g., Pitch changes when tipping forward).

ConceptNullOp.java

Description:

A minimal template for an OpMode.

Key Features:

- **Structure:** Shows the basic structure of an `OpMode` (init, loop, stop) without any functional code.

Code Breakdown:

- **Standard Methods:**

- `init()` : Runs once when INIT is pressed.
- `init_loop()` : Runs repeatedly during INIT.
- `start()` : Runs once when PLAY is pressed.
- `loop()` : Runs repeatedly during PLAY.
- `stop()` : Runs once when STOP is pressed.

- **Timer:**

- `ElapsedTime runtime` : A simple timer to show how long the OpMode has been running.

ConceptTelemetry.java

Description:

Showcases advanced telemetry features.

Key Features:

- **Rich Data:** Displaying formatted text, logs, and numeric data.
- **Lazy Evaluation:** Using `Func<T>` to compute expensive data (like battery voltage) only when needed for transmission.
- **Logging:** Scrolling text log (like a console output).

Code Breakdown:

- **Log Configuration:**

- `telemetry.log().setDisplayOrder(OLDEST_FIRST)` : Sets the order of log messages.

- `telemetry.log().setCapacity(6)` : Limits the log to 6 lines.

- **Lazy Data (`Func`):**

- `telemetry.addData(..., new Func<Double>() { ... })` : The code inside `value()` (getting battery voltage) runs ONLY when the telemetry packet is actually being built for transmission, saving CPU time if updates are throttled.

- **Rich Content:**

- `emitPoemLine()` : Adds lines to the log.

- `telemetry.addLine(...)` : Adds a text-only line, which can then have `.addData()` appended to it for a tabular look.