# Precise 360° Servo Control on ESP32 (Parallax Feedback 360°)

## Overview of the Parallax Feedback 360° Servo

The Parallax *Feedback 360° High-Speed Servo* is a continuous-rotation servo with an internal Hall-effect sensor that provides a feedback signal for its angular position. Unlike a standard servo (which is limited to ~180°), this servo can rotate continuously, and the feedback allows an external controller to close the loop and hold **any angle with unlimited range**[[1]](https://www.pololu.com/product/3432#:~:text=just%20like%20a%20typical%20continuous,4%C2%A0V)[[2]](https://www.pololu.com/product/3432#:~:text=What%20sets%20this%20servo%20apart,programming%20walkthrough%20with%20snippets%20of). The servo’s control input is a standard 50 Hz RC PWM (approx. 1280–1720 µs pulse range for full speed in each direction, with 1500 µs as stop)[[3]](https://www.pololu.com/product/3432#:~:text=the%20Feedback%20360%C2%B0%20is%20controlled,as%20the%20pulse%20width%20decreases). The feedback output is a 3.3 V PWM at about 910 Hz (period ~1.1 ms) whose **duty cycle corresponds to the servo’s current shaft angle** over a 360° rotation[[2]](https://www.pololu.com/product/3432#:~:text=What%20sets%20this%20servo%20apart,programming%20walkthrough%20with%20snippets%20of). By reading this feedback signal, you can implement a closed-loop position controller that moves the servo to a target angle (and even track multiple turns) with high precision. In fact, the feedback’s resolution is about 1024 steps per 360° (duty 2.7%–97.1%), which is roughly 0.35° per step, so achieving ~0.5° positioning accuracy is feasible with proper calibration and control[[4]](https://robotic.tistory.com/10#:~:text=int%20_dcMin%20%3D%2030%3B%20int,unitsFC%20%3D%20360%3B%20float%20_theta)[[5]](https://www.pololu.com/product/3432#:~:text=position%20feedback,Speed%20Servo).

## Closed-Loop Position Control Using the Feedback Signal

To control the servo to within 0.5° of a desired angle, you must use the servo’s feedback in a **closed-loop** fashion (open-loop timing pulses alone won’t guarantee precise angles). A proven approach is to use an **interrupt** on the feedback line to measure the high-pulse width of its 910 Hz signal in real time. For example, one implementation sets up a change interrupt on the feedback pin and records the timestamp when the signal goes HIGH and LOW. The difference gives the pulse high duration (which ranges from roughly 30 µs to 1070+ µs over 0–360°)[[6]](https://robotic.tistory.com/10#:~:text=unsigned%20long%20_dc%3B%20volatile%20unsigned,unsigned%20long%20_pulseInTimeEnd%20%3D%20micros)[[4]](https://robotic.tistory.com/10#:~:text=int%20_dcMin%20%3D%2030%3B%20int,unitsFC%20%3D%20360%3B%20float%20_theta). By ignoring any out-of-bounds readings (e.g. pulses <0 or >1500 µs, which are just noise) and using the measured high time, the code can compute the current angle. Typically this involves mapping the pulse width to a 0–359° scale. For instance, Parallax’s datasheet suggests using the measured pulse \_dc in µs with calibration constants (e.g. \_dcMin ≈ 30, \_dcMax ≈ 1071) to compute an angle:

This yields an angle θ in degrees (0–359) corresponding to the servo’s position[[4]](https://robotic.tistory.com/10#:~:text=int%20_dcMin%20%3D%2030%3B%20int,unitsFC%20%3D%20360%3B%20float%20_theta). In practice, you also maintain a **rotation counter** to handle positions beyond one revolution. Whenever the angle reading “wraps around” (e.g. from ~359° back to ~0° or vice versa), you increment or decrement a turn counter. This way you can command and track multi-turn rotations if needed. The example code by Choi uses quadrant thresholds (e.g. if the angle jumps from >270° to <90°, it assumes a full turn has passed) to update the turn count[[7]](https://robotic.tistory.com/10#:~:text=float%20getAngle%28%29,1). The absolute angle can then be calculated as angle = turns\*360 + θ (accounting for negative turns if reversed)[[8][9]](https://robotic.tistory.com/10#:~:text=if%28%28_theta%20,_q3max%29%29%20_turns).

With the current angle known on each feedback pulse (approx. 1000 updates per second), a **PID control loop** can adjust the servo’s input PWM to reduce the error between the target angle and current angle. In other words, the code will modulate the control signal around the 1500 µs “neutral” point: if the servo needs to turn clockwise to reach the target, you send a slightly shorter pulse (<1500 µs), or if it needs to go counter-clockwise, send a longer pulse (>1500 µs). As it approaches the setpoint, the PID (or simpler proportional control) will command smaller deviations from 1500 µs to gently nudge the servo into position. This closed-loop method essentially turns the continuous rotation servo into a high-precision positioning servo. For example, Choi’s demo code implements a PID loop (with tuned gains) and even adds a small offset to overcome the servo’s built-in 20 µs deadband around the stop pulse. The PID computes an output in the range -90 to +90 (interpreted as how far to offset the 90°/1500 µs neutral point). The code then writes servo.write(output + 90 + offset) to the ESP32’s PWM driver, where 90 corresponds to the 1500 µs stop command on a 0–180 scale, and offset (~8 units) recenters the output to compensate deadband[[10]](https://robotic.tistory.com/10#:~:text=void%20gotoAngle%28double%20targetAngle%29%20,deg%2Fs%20_timeP%20%3D%20_time)[[11]](https://robotic.tistory.com/10#:~:text=_angleP%20%3D%20_angle%3B). Using this approach, the servo can be made to **turn to and hold a specified angle** with minimal steady-state error[[1]](https://www.pololu.com/product/3432#:~:text=just%20like%20a%20typical%20continuous,4%C2%A0V)[[2]](https://www.pololu.com/product/3432#:~:text=What%20sets%20this%20servo%20apart,programming%20walkthrough%20with%20snippets%20of). In Choi’s example, they command the servo to rotate **five full turns** forward and back, and the closed-loop controller holds the desired position each time within a small error tolerance[[12]](https://robotic.tistory.com/10#:~:text=%7B%20goTo%28360%20,360%20%2A%205%29%3B%20delay%285000%29%3B)[[13]](https://robotic.tistory.com/10#:~:text=while%28%28millis%28%29,write%2890%29%3B). This demonstrates that with feedback and a fast control loop, even continuous servos can achieve reliable position accuracy.

## Proven Code and Libraries for the Parallax 360 on ESP32

Fortunately, you don’t have to write all of this from scratch – there are **existing libraries and code examples** specifically made for the Parallax Feedback 360 servo:

* **ESP32Servo360 Library (Arduino/PlatformIO)** – This open-source library is designed for ESP32 boards and the Parallax 360 servo[[14]](https://github.com/ecal-mid/ESP32Servo360#:~:text=ESP32Servo360). It uses interrupts to read the feedback and runs the control loop (leveraging ESP32’s multitasking capabilities for smooth operation). You can install it via the Arduino Library Manager or PlatformIO. Using it is straightforward: you create an ESP32Servo360 servo; object and attach it to the appropriate pins. For example: servo.attach(controlPin, feedbackPin). The library allows you to set parameters like offset angle, maximum RPM, and it has built-in easing for smooth deceleration near the target[[15]](https://github.com/ecal-mid/ESP32Servo360#:~:text=,Automatic%20by%20default)[[16]](https://github.com/ecal-mid/ESP32Servo360#:~:text=). To move the servo, you simply call servo.setTarget(angle), and the library handles driving the motor to that absolute angle and holding it. The documentation notes that by default it assumes the feedback pulse spans ~32 (0°) to 1067 (360°) timer counts, and it even provides a calibration function to refine these values for maximum accuracy[[17]](https://github.com/ecal-mid/ESP32Servo360#:~:text=,Automatic%20by%20default)[[18]](https://github.com/ecal-mid/ESP32Servo360#:~:text=,servo.calibrate). In other words, *ESP32Servo360* was created to do exactly what you need – closed-loop position control of the Parallax 360 – with minimal fuss. *(After installing, check the examples under* *File > Examples > ESP32Servo360* *for a template.)*
* **FeedBackServo Arduino Library** – Another proven solution is the *FeedBackServo* library by HyodaKazuaki[[19]](https://forum.arduino.cc/t/parallax-feedback-360-servo-control/676831/7#:~:text=At%20any%20rate%2C%20I%20decided,360%C2%B0%20High%20Speed%20Servo%20easy)[[20]](https://forum.arduino.cc/t/parallax-feedback-360-servo-control/676831/7#:~:text=,define%20SERVO_PIN%203). This library works on Arduino-compatible boards (the author tested on Uno/Mega, but it can be adapted to ESP32 as well, since it uses generic attachInterrupt). To use it, include **FeedBackServo.h** in your project. You instantiate a servo object with the feedback pin number, then call servo.setServoControl(controlPin) to specify the control signal pin[[21]](https://github.com/HyodaKazuaki/Parallax-FeedBack-360-Servo-Control-Library-4-Arduino#:~:text=). The library internally starts interrupts on the feedback pin to constantly update the servo’s angle. You can then set a target in degrees and let the library drive the servo towards it. For example, you can do: servo.setKp(1.0); to set a proportional gain, and servo.setTarget(90); to move to 90°[[22]](https://github.com/HyodaKazuaki/Parallax-FeedBack-360-Servo-Control-Library-4-Arduino#:~:text=servo)[[23]](https://github.com/HyodaKazuaki/Parallax-FeedBack-360-Servo-Control-Library-4-Arduino#:~:text=case%200%3A%20target%20%3D%201%3B,setTarget%28180%29%3B%20break%3B). In your main loop, calling servo.update(tolerance) will update the control output and stop when the error is within the given tolerance (in degrees)[[24]](https://github.com/HyodaKazuaki/Parallax-FeedBack-360-Servo-Control-Library-4-Arduino#:~:text=%2F%2F%20Rotate%20servo%20from%200,blocking.%20servo.update%282). The library’s example code demonstrates sweeping to specific angles: e.g. servo.rotate(270, 4); moves to 270° (with ±4° allowed error) and holds, then servo.rotate(-180, 4); moves to –180°[[25]](https://forum.arduino.cc/t/parallax-feedback-360-servo-control/676831/7#:~:text=void%20loop%28%29%20,rotate%28270%2C%204%29%3B%20delay%282000). Under the hood, this library implements the feedback reading and a simple PID (primarily P-control via the Kp setting) to adjust the servo’s speed until the target angle is reached. Many users have reported success with this library on Arduinos to achieve precise angle control of the Parallax 360. If using it on an ESP32 (via PlatformIO with the Arduino framework), ensure the chosen feedback GPIO is capable of interrupts and not used by other functions.
* **Custom PID Control (DIY Approach)** – For maximum control (or if you’re using ESP-IDF), you can write your own feedback loop code. Parallax provides a low-level C example (for their Propeller microcontroller) in the servo’s product guide, which can be adapted to other platforms[[26]](https://www.pololu.com/file/0J1395/900-00360-Feedback-360-HS-Servo-v1.2.pdf#:~:text=C%20example%20code%20for%20low,should%20provide%20an%20illustration%20of). The core idea is what we described: use a hardware timer or interrupt to measure the feedback pulse, compute the angle, then adjust the PWM output accordingly. The blog post by H.J. Choi offers a great reference implementation in C++ for the ESP32, including a PID loop and rotation counting logic[[27]](https://robotic.tistory.com/10#:~:text=I%20used%20a%20,interrupt%20function%20looks%20like%20below)[[10]](https://robotic.tistory.com/10#:~:text=void%20gotoAngle%28double%20targetAngle%29%20,deg%2Fs%20_timeP%20%3D%20_time). In Choi’s code, a class parallexServo (sic) encapsulates the functionality – in the constructor it sets up the LEDC PWM for the servo output (configuring 50 Hz and appropriate min/max pulse widths) and attaches the feedback interrupt[[28]](https://robotic.tistory.com/10#:~:text=public%3A%20parallexServo,feedPin%3B%20_outPin%20%3D%20outPin)[[29]](https://robotic.tistory.com/10#:~:text=void%20signalInterrupt%28%29%20,dc%20%3D%20temp%3B%7D). The signalInterrupt() ISR captures the pulse width, and getAngle() returns the current multi-turn angle reading in degrees[[30]](https://robotic.tistory.com/10#:~:text=_theta%20%3D%20%28_unitsFC%20,1)[[9]](https://robotic.tistory.com/10#:~:text=if%28%28_theta%20,_q3max%29%29%20_turns). A gotoAngle(targetAngle) method runs the PID computations and continually drives the servo toward the target, including a small “offset” to overcome the 1–2µs deadband around the stop command[[10]](https://robotic.tistory.com/10#:~:text=void%20gotoAngle%28double%20targetAngle%29%20,deg%2Fs%20_timeP%20%3D%20_time)[[11]](https://robotic.tistory.com/10#:~:text=_angleP%20%3D%20_angle%3B). This code reportedly achieved smooth and accurate control; the example usage shows commanding the servo to spin 5 rotations forward and back with a stop error of only ~2 degrees in the loop (which could be tightened further with tuning)[[13]](https://robotic.tistory.com/10#:~:text=while%28%28millis%28%29,write%2890%29%3B)[[12]](https://robotic.tistory.com/10#:~:text=%7B%20goTo%28360%20,360%20%2A%205%29%3B%20delay%285000%29%3B). Adapting this approach in your own project is an option if you need a tailored solution with the *lowest possible latency* (e.g. using ESP32’s MCPWM or RMT modules for capturing the feedback signal might further reduce timing jitter). But in most cases, the ready-made libraries above are sufficient and have been **proven to work with the Parallax Feedback 360° servo** out of the box.

## Key Takeaways

* **Use the Servo’s Feedback:** Achieving ~0.5° precision requires closing the loop with the servo’s built-in Hall sensor feedback. The 910 Hz PWM feedback gives you the current angle, which you can trust for high-resolution positioning[[2]](https://www.pololu.com/product/3432#:~:text=What%20sets%20this%20servo%20apart,programming%20walkthrough%20with%20snippets%20of). Simply driving the servo open-loop (with fixed pulses) won’t guarantee that level of accuracy, due to load variation and the servo’s internal deadband.
* **Leverage Existing Code:** There are dedicated libraries like *ESP32Servo360* for ESP32[[14]](https://github.com/ecal-mid/ESP32Servo360#:~:text=ESP32Servo360) and *FeedBackServo* for Arduino[[19]](https://forum.arduino.cc/t/parallax-feedback-360-servo-control/676831/7#:~:text=At%20any%20rate%2C%20I%20decided,360%C2%B0%20High%20Speed%20Servo%20easy) which handle the heavy lifting. They use interrupts for low-latency feedback capture and adjust the servo’s PWM input continuously to reach the target angle. By using these, you can avoid reinventing the control algorithm and get straight to tuning your system (e.g. adjusting the Kp or target speeds).
* **Tune and Calibrate:** To get the best precision, calibrate the feedback timing to your specific servo unit (the *ESP32Servo360* library provides a calibrate() function to measure the exact min/max duty counts[[18]](https://github.com/ecal-mid/ESP32Servo360#:~:text=,servo.calibrate)). Also, you might need to tune the PID or proportional gain. A too-high gain can cause oscillation (as some users observed oscillation or jitter when holding position[[31]](https://forum.arduino.cc/t/parallax-feedback-360-servo-control/676831/7#:~:text=Problem%202%3A%20Code%20like%20below%2C,back%20and%20forth%205%20degrees)), while too low a gain might make the servo sluggish. Start with modest values (the examples use Kp ≈ 0.5–1.0)[[32]](https://github.com/HyodaKazuaki/Parallax-FeedBack-360-Servo-Control-Library-4-Arduino#:~:text=servo) and adjust as needed to get a snappy but stable response.
* **Power and Wiring:** Don’t overlook the hardware setup. Use a stable 6 V supply capable of providing up to ~1.2 A for the servo, especially under load. Connect the servo’s **yellow feedback wire** to a GPIO on the ESP32 that supports interrupts, the **white control wire** to a PWM-capable GPIO for output, and common ground between the ESP32 and servo power[[33]](https://github.com/ecal-mid/ESP32Servo360#:~:text=Image%3A%20wiring). Many ESP32 boards use 3.3 V logic which is perfect for the feedback signal (no level shifting needed). If using the servo’s full speed (up to 140 RPM), consider ramping down as you approach the target to minimize overshoot – the libraries often include such deceleration features by default[[34]](https://github.com/ecal-mid/ESP32Servo360#:~:text=,Automatic%20by%20default)[[35]](https://github.com/ecal-mid/ESP32Servo360#:~:text=).

By using the above resources, you should be able to precisely control the Parallax Feedback 360° servo with your ESP32 (ESP32-WROOM-32) in a closed-loop manner. In summary, **yes – people have figured this out**, and you can leverage their code. With the *feedback PWM* as your guide and a well-tuned control loop, your robotic project’s servo can be positioned within about half a degree of accuracy reliably[[2]](https://www.pololu.com/product/3432#:~:text=What%20sets%20this%20servo%20apart,programming%20walkthrough%20with%20snippets%20of)[[36]](https://robotic.tistory.com/10#:~:text=Continuous%20servo%20motors%20also%20use,motor%20in%20small%20wheeled%20robots). Good luck with your implementation!

**Sources:** Parallax/Pololu product documentation[[2]](https://www.pololu.com/product/3432#:~:text=What%20sets%20this%20servo%20apart,programming%20walkthrough%20with%20snippets%20of)[[3]](https://www.pololu.com/product/3432#:~:text=the%20Feedback%20360%C2%B0%20is%20controlled,as%20the%20pulse%20width%20decreases); H.J. Choi’s ESP32 PID control example[[36]](https://robotic.tistory.com/10#:~:text=Continuous%20servo%20motors%20also%20use,motor%20in%20small%20wheeled%20robots)[[10]](https://robotic.tistory.com/10#:~:text=void%20gotoAngle%28double%20targetAngle%29%20,deg%2Fs%20_timeP%20%3D%20_time); *ESP32Servo360* library README[[14]](https://github.com/ecal-mid/ESP32Servo360#:~:text=ESP32Servo360)[[17]](https://github.com/ecal-mid/ESP32Servo360#:~:text=,Automatic%20by%20default); HyodaKazuaki’s *FeedBackServo* library and examples[[32]](https://github.com/HyodaKazuaki/Parallax-FeedBack-360-Servo-Control-Library-4-Arduino#:~:text=servo)[[25]](https://forum.arduino.cc/t/parallax-feedback-360-servo-control/676831/7#:~:text=void%20loop%28%29%20,rotate%28270%2C%204%29%3B%20delay%282000).

[[1]](https://www.pololu.com/product/3432#:~:text=just%20like%20a%20typical%20continuous,4%C2%A0V) [[2]](https://www.pololu.com/product/3432#:~:text=What%20sets%20this%20servo%20apart,programming%20walkthrough%20with%20snippets%20of) [[3]](https://www.pololu.com/product/3432#:~:text=the%20Feedback%20360%C2%B0%20is%20controlled,as%20the%20pulse%20width%20decreases) [[5]](https://www.pololu.com/product/3432#:~:text=position%20feedback,Speed%20Servo) Pololu - Parallax Feedback 360° High-Speed Servo

<https://www.pololu.com/product/3432>

[[4]](https://robotic.tistory.com/10#:~:text=int%20_dcMin%20%3D%2030%3B%20int,unitsFC%20%3D%20360%3B%20float%20_theta) [[6]](https://robotic.tistory.com/10#:~:text=unsigned%20long%20_dc%3B%20volatile%20unsigned,unsigned%20long%20_pulseInTimeEnd%20%3D%20micros) [[7]](https://robotic.tistory.com/10#:~:text=float%20getAngle%28%29,1) [[8]](https://robotic.tistory.com/10#:~:text=if%28%28_theta%20,_q3max%29%29%20_turns) [[9]](https://robotic.tistory.com/10#:~:text=if%28%28_theta%20,_q3max%29%29%20_turns) [[10]](https://robotic.tistory.com/10#:~:text=void%20gotoAngle%28double%20targetAngle%29%20,deg%2Fs%20_timeP%20%3D%20_time) [[11]](https://robotic.tistory.com/10#:~:text=_angleP%20%3D%20_angle%3B) [[12]](https://robotic.tistory.com/10#:~:text=%7B%20goTo%28360%20,360%20%2A%205%29%3B%20delay%285000%29%3B) [[13]](https://robotic.tistory.com/10#:~:text=while%28%28millis%28%29,write%2890%29%3B) [[27]](https://robotic.tistory.com/10#:~:text=I%20used%20a%20,interrupt%20function%20looks%20like%20below) [[28]](https://robotic.tistory.com/10#:~:text=public%3A%20parallexServo,feedPin%3B%20_outPin%20%3D%20outPin) [[29]](https://robotic.tistory.com/10#:~:text=void%20signalInterrupt%28%29%20,dc%20%3D%20temp%3B%7D) [[30]](https://robotic.tistory.com/10#:~:text=_theta%20%3D%20%28_unitsFC%20,1) [[36]](https://robotic.tistory.com/10#:~:text=Continuous%20servo%20motors%20also%20use,motor%20in%20small%20wheeled%20robots) Continuous Servo Control for ESP32

<https://robotic.tistory.com/10>

[[14]](https://github.com/ecal-mid/ESP32Servo360#:~:text=ESP32Servo360) [[15]](https://github.com/ecal-mid/ESP32Servo360#:~:text=,Automatic%20by%20default) [[16]](https://github.com/ecal-mid/ESP32Servo360#:~:text=) [[17]](https://github.com/ecal-mid/ESP32Servo360#:~:text=,Automatic%20by%20default) [[18]](https://github.com/ecal-mid/ESP32Servo360#:~:text=,servo.calibrate) [[33]](https://github.com/ecal-mid/ESP32Servo360#:~:text=Image%3A%20wiring) [[34]](https://github.com/ecal-mid/ESP32Servo360#:~:text=,Automatic%20by%20default) [[35]](https://github.com/ecal-mid/ESP32Servo360#:~:text=) GitHub - ecal-mid/ESP32Servo360: ESP32 Library to control Parallax High Speed 360° servo motors.

<https://github.com/ecal-mid/ESP32Servo360>

[[19]](https://forum.arduino.cc/t/parallax-feedback-360-servo-control/676831/7#:~:text=At%20any%20rate%2C%20I%20decided,360%C2%B0%20High%20Speed%20Servo%20easy) [[20]](https://forum.arduino.cc/t/parallax-feedback-360-servo-control/676831/7#:~:text=,define%20SERVO_PIN%203) [[25]](https://forum.arduino.cc/t/parallax-feedback-360-servo-control/676831/7#:~:text=void%20loop%28%29%20,rotate%28270%2C%204%29%3B%20delay%282000) [[31]](https://forum.arduino.cc/t/parallax-feedback-360-servo-control/676831/7#:~:text=Problem%202%3A%20Code%20like%20below%2C,back%20and%20forth%205%20degrees) Parallax feedback 360 Servo control - #7 by tamaren - Programming - Arduino Forum

<https://forum.arduino.cc/t/parallax-feedback-360-servo-control/676831/7>

[[21]](https://github.com/HyodaKazuaki/Parallax-FeedBack-360-Servo-Control-Library-4-Arduino#:~:text=) [[22]](https://github.com/HyodaKazuaki/Parallax-FeedBack-360-Servo-Control-Library-4-Arduino#:~:text=servo) [[23]](https://github.com/HyodaKazuaki/Parallax-FeedBack-360-Servo-Control-Library-4-Arduino#:~:text=case%200%3A%20target%20%3D%201%3B,setTarget%28180%29%3B%20break%3B) [[24]](https://github.com/HyodaKazuaki/Parallax-FeedBack-360-Servo-Control-Library-4-Arduino#:~:text=%2F%2F%20Rotate%20servo%20from%200,blocking.%20servo.update%282) [[32]](https://github.com/HyodaKazuaki/Parallax-FeedBack-360-Servo-Control-Library-4-Arduino#:~:text=servo) GitHub - HyodaKazuaki/Parallax-FeedBack-360-Servo-Control-Library-4-Arduino: Arduino library which control Parallax FeedBack 360° High Speed Servo easy.

<https://github.com/HyodaKazuaki/Parallax-FeedBack-360-Servo-Control-Library-4-Arduino>

[[26]](https://www.pololu.com/file/0J1395/900-00360-Feedback-360-HS-Servo-v1.2.pdf#:~:text=C%20example%20code%20for%20low,should%20provide%20an%20illustration%20of) Parallax Feedback 360° High-Speed Servo (#900-00360) Product Guide

<https://www.pololu.com/file/0J1395/900-00360-Feedback-360-HS-Servo-v1.2.pdf>