



Pinewood Derby Electronics

Setup & Entity Mapping | Unit 67 | v1.1

Install the Home Assistant police-car card, connect its entities, and prepare the ESPHome reference. The build uses a Sseed Studio XIAO ESP32C3, MPU6050 motion sensor, and a rechargeable 3.7 V, 650 mAh LiPo battery.

1. Get the project files

Download the firmware reference, cleaned dashboard card, and graphics ZIP from the [Pinewood Derby project page](#).

File	Purpose
pine-esphome-reference.yaml	Public ESPHome firmware reference; local secret values required.
pine-dashboard-card.yaml	Home Assistant custom:button-card configuration.
pinewood-dashboard-assets.zip	Matching police-car and city SVG graphics.

2. Install the card and graphics

Install **custom:button-card** in Home Assistant. Extract the graphics ZIP into **/config/www/**, keeping its pinewood folder. The two final file paths are:

```
/config/www/pinewood/pinewood-police-car.svg
```

```
/config/www/pinewood/pinewood-city.svg
```

Home Assistant serves these files as **/local/pinewood/...**. The downloaded card already uses those paths. Add a Manual card to your dashboard, then paste the contents of **pine-dashboard-card.yaml**.

3. Prepare private ESPHome secrets

Keep your own credentials in local **secrets.yaml**. The public firmware references these five keys; it does not include their values:

```
pine_api_encryption_key, pine_ota_password, pine_fallback_password, wifi_ssid, wifi_password
```

The supplied firmware target is **esp32-c3-devkitm-1**; its original logic and target are retained. Review the target and raw GPIO assignments for your XIAO before flashing. This download is a reference, not a compiled hardware-tested release.



Map entities & calibrate

4. Match the entities in your installation

Find the car under Home Assistant **Settings > Devices & services > ESPHome**. Copy each actual entity ID and replace its example below throughout the card, including **triggers_update**, the main **entity**, and JavaScript **states[...]** lookups. IDs can differ after device renaming.

Reading / state	Example entity ID
Front / back tilt	sensor.pine_front_back_tilt
Left / right tilt	sensor.pine_left_right_tilt
Battery voltage (V)	sensor.pinewood_battery_voltage
Battery level (%)	sensor.pine_battery_level
MPU chip temperature	sensor.pine_mpu_temperature
Car level	binary_sensor.pine_car_level
Car active	binary_sensor.pine_car_active
Optional raw Z acceleration	sensor.pine_accel_z

The supplied card displays battery volts; Battery Level is an additional firmware sensor for other dashboard layouts. Its ID above is an example. The **LEVEL** badge describes car tilt, not battery charge.

The active entity controls sleep display; missing tilt shows **UNAVAILABLE**. The optional raw-Z upside-down effect depends on mounting orientation.

5. Calibrate during one awake session

1. Hold the car flat and press **Capture Flat**.
2. Lift the front, hold steady, and press **Capture Front Lift**.
3. Lift one side, hold steady, and press **Capture Side Lift**.
4. Press **Solve Car Calibration** to save the car axes.
5. Return to a level surface and press **Zero Car Level**.

Complete all three captures before the car sleeps. Lift captures are temporary; solved axes and zero offsets are saved. Both angles within +/-2 degrees indicate level.

6. Check the finished dashboard

Battery input: GPIO4 = XIAO D2/A2 (ADC1). The external monitor must supply half the battery voltage; the firmware multiplies this ADC reading by 2. BAT+ alone does not provide a voltage reading. Compare Battery Voltage with a multimeter while awake.

Battery Level is an estimate from the same reading: 3.55 V or lower = 0%, 4.10 V or higher = 100%, with a linear estimate between. It needs no second pin. Check tilt direction, temperature and wake/sleep behavior on the assembled car.

The website demonstration uses sample readings and does not connect to your car. For help, visit tryonelectronics.com/support.