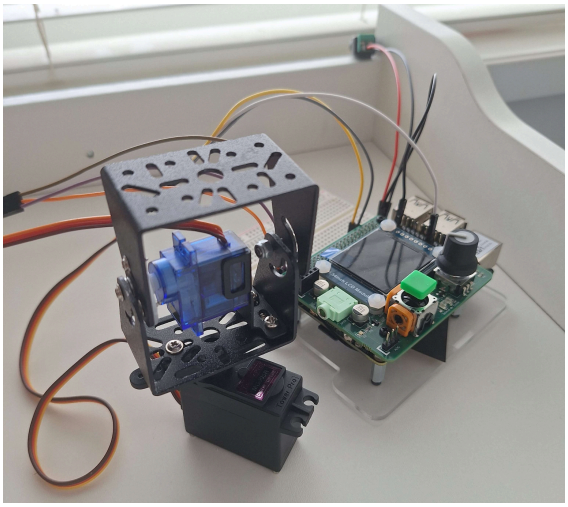
Pan / Tilt Kit Guide for BeagleY-AI

CMPT 433: Embedded Systems
Andre Luong, Brian Le, Dongan Kim
Spring 2025



Guide has been tested on:

- BeagleY-AI (Target): Debian 12.x
- PC OS (Host): Debian 12.x

Equipment:

- BeagleY-AI
- Dagu Mini Pan/Tilt Kit
- Breadboard
- 9 Wires (6 M-M, 3 M-F)
- External Power Supply (5V, 1+ Amps) with a 2.1mm DC Power Adapter

Note:

- A servo from our pan/tilt kit was defective, so we replaced it with another servo

Table of Contents:

Introduction

This guide introduces how to integrate a pan/tilt kit with a BeagleY-AI board and control it using a simple C++ program.

The kit uses two servos which are controlled using pulse-width modulation (PWM). We will use a PWM period of 50,000,000 ns (50 ms). This produces a slow, but smooth turn when changing angles. The duty cycle will range from 500,000 to 2,500,000 ns (0.5 to 2.5 ms), corresponding to the servo's maximum rotation of 180°.

Pulse-Width Modulation

BeagleY-AI provides a few PWM modules to use. For this setup:

Pin #	Pin Name	PWM Module	Usage	Overlay
8	GPIO14	PWM0	Tilt	k3-am67a-beagley-ai-pwm-epwm0-gpio14.dtbo
31	GPIO6	PWM1	Pan	k3-am67a-beagley-ai-pwm-epwm1-gpio6.dtbo

Enable these PWM pins by editing the */boot/firmware/extlinux/extlinux.conf* file and loading each overlay. Reboot your board afterwards.

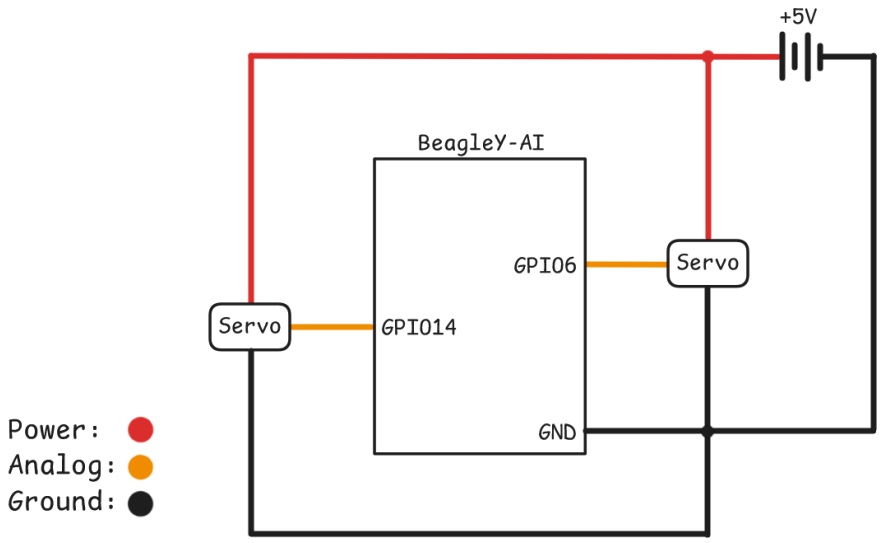
Before each use, configure the HAT pin PWM symlink pin:

```
(target)$ sudo beagle-pwm-export --pin hat-08
(target)$ sudo beagle-pwm-export --pin hat-31
```

It's recommended to include these commands in a script to avoid excessive user configuration.

For more details on configuring PWM, refer to the course's PWM Guide.

Circuit Wiring

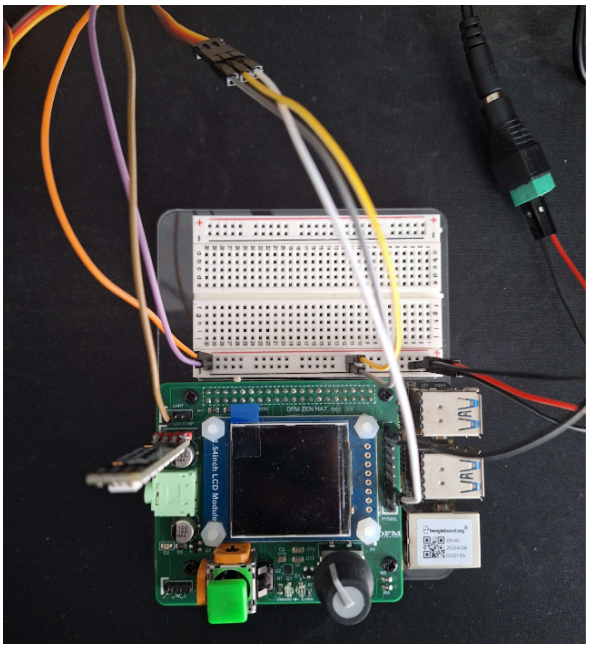


Given that [each servo consumes up to 500mA], we required an external power supply to provide the 1A that the BeagleY-AI board can't sufficiently supply.

Each servo has a wire attached with 3 colours: red for power, brown for ground, and yellow/orange for analog signal.

Note that no resistors are needed, as the servos can accept the analog signal directly from the GPIO pins.

1. Attach 2 M-M wires to the DC Power Adapter. Use a screwdriver to unlock and lock the wire ends in place. Attach the other ends to the power rails of the breadboard.
2. For each servo:
    a. Attach 2 M-M wires for power and ground to the power rails.
    b. Attach a M-F wire for analog to a GPIO pin. PYMNL 3 has GPIO6 and UART 1 has GPIO14.
3. Attach a M-F wire from PYMNL 10 (GND) to the ground rail of the breadboard. This would ground the entire circuit.
4. Connect the DC Power Adapter with the Power Supply Adapter.



Power Supply
Red: pwr
Black: gnd

Pan:
Yellow: pwr
Grey: gnd
White: gpio6

Tilt:
Orange: pwr
Purple: gnd
Brown: gpio14

PYMNL 10:
black: gnd

Controlling with C++

Once the circuit has been properly configured and connected, we can run the following C++ program to test the pan/tilt kit functionality. This program simply turns the pan, left-right-left, and tilt, down-up-down.

```
Libraries:
#include <iostream>
#include <fstream>
#include <unistd.h>
```

```
Variables:
const std::string& PAN_FILE_PATH = "/dev/hat/pwm/GPIO6/";
const std::string& TILT_FILE_PATH = "/dev/hat/pwm/GPIO14/";

const std::string& PERIOD = "period";
const std::string& DUTY_CYCLE = "duty_cycle";
const std::string& ENABLE = "enable";

constexpr int ONE_SECOND_NS = 1000000000;
constexpr int MIN_DUTY_CYCLE = 500000;  // 0.5ms pulse width
constexpr int MAX_DUTY_CYCLE = 2500000; // 2.5ms pulse width
constexpr int MAX_ROTATION = 180;
```

Functions:

```
// Write to Period, Duty Cycle, and Enable files with value
static void writeToFile(const std::string& filePath, const std::string& file, const
std::string& value) {
	std::ofstream out;
	out.open(filePath + file);
	if (!out) {
		std::cerr << "[Error] Can't write " << value << " to file: "
			<< filePath << file << std::endl;
		return;
	}
	out << value;
	out.close();
}

static void setEnable(const std::string& filePath, const int& value) {
	writeToFile(filePath, ENABLE, std::to_string(value));
}

static void setDutyCycle(const std::string& filePath, const int& value) {
	writeToFile(filePath, DUTY_CYCLE, std::to_string(value));
}

static void setPeriod(const std::string& filePath, const int& hertz) {
	auto frequencyNs = ONE_SECOND_NS / hertz;
	writeToFile(filePath, PERIOD, std::to_string(frequencyNs));
}

static void setServoAngle(const std::string& filePath, const int& angle) {
	// Bounds angle to [MIN_DUTY_CYCLE, MAX_DUTY_CYCLE]
	int dutyCycle = MIN_DUTY_CYCLE + ((MAX_DUTY_CYCLE - MIN_DUTY_CYCLE) * angle)
			/ MAX_ROTATION;
	setDutyCycle(filePath, dutyCycle);

	// Give time to run
	sleep(1);
}
```

Test Program:

```
int main() {
	// Initialize
	setDutyCycle(PAN_FILE_PATH, 0);
	setDutyCycle(TILT_FILE_PATH, 0);
	setPeriod(PAN_FILE_PATH, 20);
```

```
        setPeriod(TILT_FILE_PATH, 20);
        setEnable(PAN_FILE_PATH, 1);
        setEnable(TILT_FILE_PATH, 1);

        printf("Starting...\n");
        sleep(1);

        // Change servo angles
        setServoAngle(PAN_FILE_PATH, 0);
        setServoAngle(TILT_FILE_PATH, 0);

        setServoAngle(PAN_FILE_PATH, 180);
        setServoAngle(TILT_FILE_PATH, 180);

        setServoAngle(PAN_FILE_PATH, 0);
        setServoAngle(TILT_FILE_PATH, 0);

        // Disable
        setEnable(PAN_FILE_PATH, 0);
        setEnable(TILT_FILE_PATH, 0);

        printf("Done!\n");
        return 0;
}
```

Troubleshooting

Unknown pin name: [--pin hat-#]

- The HAT pin PWM symlink pin has not been configured correctly. Double check the naming in the Pulse-Width Modulation section.

[Error] Can't write <value> to file <file>

- This error comes from running the C++ program. Ensure that the correct pins are used and the overlays are added correctly. Refer to the PWM Guide for more detail on set up.

Servo is not functioning

- This may be a result of defective components. Double check the wires, servos, and power supply by swapping out parts and GPIO pins. Replace any defective components found.