Tyr Electronics Plan

Contents

Problem Statement

The current BOM for electronics is bulky and looks like it almost doesn't fit in the body. After reading comments on the YouTube video and some research on the web and with Copilot, we came up with an alternative plan that requires less parts and will more easily fit in the rover body.

Parts

6 - JGA25-370 motors 6v 130rpm (6v)

6 - DS041MG 5KG 180 Servos (Operation Voltage Range: 4. 8V up to 8.4 V)

1 - XT60 to bare wire

1 – LiPo Charger

1 – use 3s battery 11.4v 2200mAh with 3 BEC's

1 for Raspberry pi, servos and other 5 volt electronics

1 for 3 left side motors, 1 for 3 right side motors

3 – USB-C Power breakout cables for Pi two motor controllers

1 - PCA9685 servo control board. Servos require 600 ma stall. Control with a PCA9685 board from Pi with I2C. Each pin can handle 2A. Takes 5-10V.

1 - Use Raspberry Pi 5 for control of robot.

1 - Grove shield for Ic2 and other sensors

2 - Motor 2040 - Quad Motor Controllers - $25. Takes 5v input. It only has .5 amp output. Motors take 1.8A stall. Power motors directly from battery. Will need some sort of power distribution from battery to motors. 2.7-10V

Use longer 6 Pin JST-SH Cable from motor to controllers and power. 520 mm or 20.5"

Motor 2040 CircuitPython

The Adafruit CircuitPython library had the best support. It supports setting the Motor 2040 as an i2c listener.

- https://learn.adafruit.com/welcome-to-circuitpython
- Download and Install CircuitPython
- Download and Install CircuitPython Libraries
    - Copy adafruit_motor folder to lib folder on Motor 2040

Motor Specification

DC12V 25GA370 Large Torque Speed Reduction Gear Motor with Tachometer Encoder Specification:

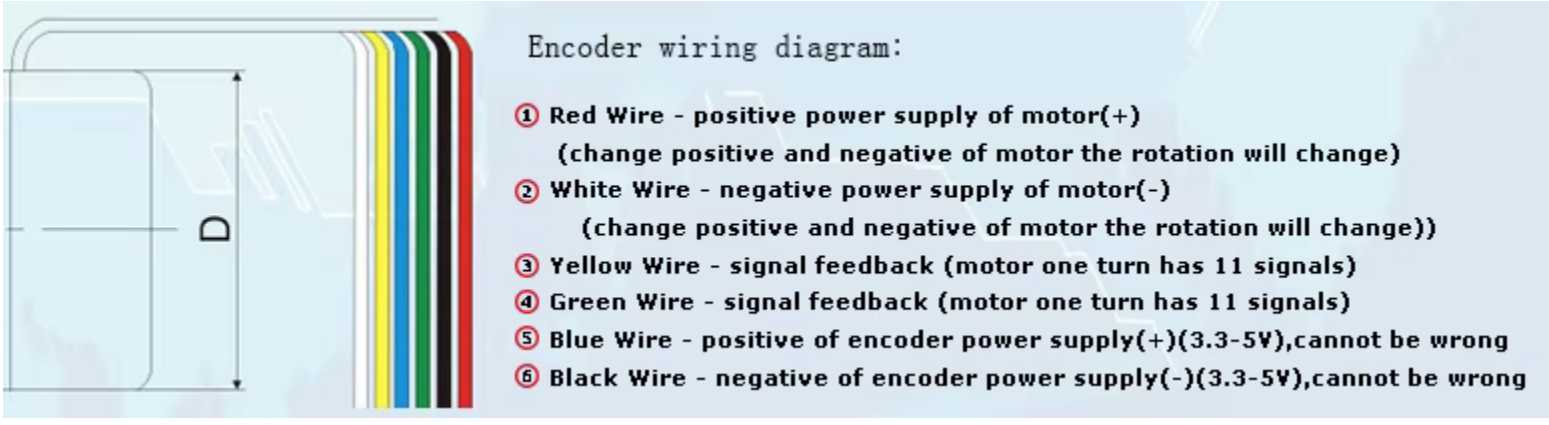
- Voltage: DC6V
- NO-Load Speed: 130RPM
- Error: ±5%
- Motor Diameter: 24.5mm

- Motor Body Length: 40mm(ref)
- Output Shaft Diameter: 4mm(D-type)
- Output Shaft Length: about 9.5mm
- Gearbox Diameter: 25mm
- Gearbox Size Detail, please see the table below

Motor and Motor 2040 Wiring

The Motor wiring is from [JGA25-370 motors 6v 130rpm]

- Red Wire - positive power supply of motor(+)(change positive and negative of motor the rotation will change)
- White Wire - negative power supply of motor(-)(change positive and negative of motor the rotation will change))
- Yellow Wire - signal feedback (motor one turn has 11 signals)
- Green Wire - signal feedback (motor one turn has 11 signals)
- Blue Wire - positive of encoder power supply(+)(3.3-5V),cannot be wrong
- Black Wire - negative of encoder power supply(-)(3.3-5V),cannot be wrong



Motor to Motor 2040 Wiring Table

Pin	Description	Motor	Motor 2040 JST-SH

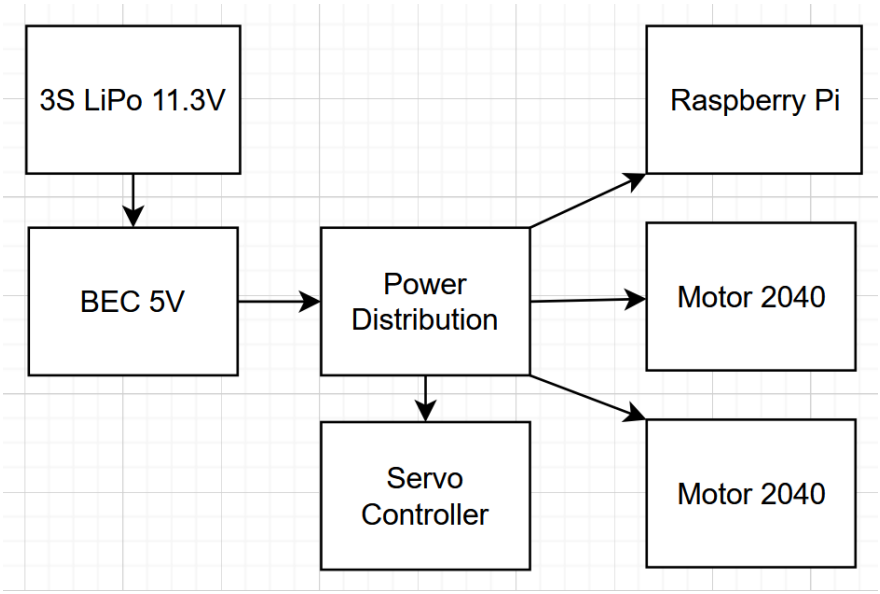
1	Motor Positive (M+)	Red pwr+	Blue out_mot-
2	Motor Negative (M-)	White pwr-	White out_mot+
3	3.3V	Yellow sig_fb	Yellow encA
4	Motor A (Encoder A)	Green sig_fb	Black encB
5	Motor B (Encoder B)	Blue enc+	Green +3.3V
6	Ground (GND)	Black enc-	Red ground

- The first pair: power from 2040 to spin the motor, can be swapped (*it is in our configuration*)
- The second pair: the feedback from the motor to 2040's encoder
- The third pair: power from 2040 to the motor's encoder

Motor specifications

Reduction ratio	Rated Volt	No Load		AT Load			STALL		Gearbox
		SPEED	CURRENT	Torque	SPEED	Current	TOGQCE	CURRENT	Length
	V	RPM	mA	KG.cm	RPM	A	KG..CM	A	mm
4.4	6	1360	100	0.1	1000	0.45	0.35	1.8	17
9.6	6	620	100	0.22	450	0.45	0.75	1.8	17
21.3	6	280	100	0.5	220	0.45	1.7	1.8	19
35	6	170	100	0.8	130	0.45	2.8	1.8	21
46	6	130	100	1	100	0.45	3.6	1.8	21
78	6	77	100	1.8	60	0.45	6.2	1.8	23
103	6	60	100	2.4	46	0.45	8.2	1.8	23
171	6	35	100	4	27	0.45	9	1.8	25
226	6	26	100	5.2	20	0.45	9	1.8	25
377	6	16	100	8.4	12	0.45	9	1.8	27
500	6	12	100	9	9	0.45	9	1.8	27
4.4	12	1360	60	0.1	1000	0.45	0.35	1.3	17
9.6	12	620	60	0.22	450	0.45	0.75	1.3	17
21.3	12	280	60	0.5	220	0.45	1.7	1.3	19
35	12	170	60	0.8	130	0.45	2.8	1.3	21
46	12	130	60	1	100	0.45	3.6	1.3	21
78	12	77	60	1.8	60	0.45	6.2	1.3	23
103	12	60	60	2.4	46	0.45	8.2	1.3	23
171	12	35	60	4	27	0.45	9	1.3	25
226	12	26	60	5.2	20	0.45	9	1.3	25
377	12	16	60	8.4	12	0.45	9	1.3	27
500	12	12	60	9	9	0.45	9	1.3	27
4.4	24	1360	40	0.1	1000	0.25	0.35	0.7	17
9.6	24	620	40	0.22	450	0.25	0.75	0.7	17
21.3	24	280	40	0.5	220	0.25	1.7	0.7	19
35	24	170	40	0.8	130	0.25	2.8	0.7	21
46	24	130	40	1	100	0.25	3.6	0.7	21
78	24	77	40	1.8	60	0.25	6.2	0.7	23
103	24	60	40	2.4	46	0.25	8.2	0.7	23
171	24	35	40	4	27	0.25	9	0.7	25
226	24	26	40	5.2	20	0.25	9	0.7	25
377	24	16	40	8.4	12	0.25	9	0.7	27
500	24	12	40	9	9	0.25	9	0.7	27

Power Distribution



1 – use 3s battery 11.4v with 3 BEC voltage converter's

1 - XT60 to bare wire

1 – LiPo Charger

3 – USB-C Power breakout cables for Pi and two motor controllers to power distribution

1 - PCA9685 servo control board Servos require 600 ma stall. Control with a PCA9685 board from Pi with I2C. Each pin can handle 2A. Takes 5-10V

1 - Use Raspberry Pi 5 Needs 5V

2 - Motor 2040 - Quad Motor Controllers - $25. Takes 5v input. It has .5 amp output per motor. Motors take 1.8A stall.

Basic Project with PWM Motor Control

Materials Needed:

1. **Motor 2040 - Quad Motor Controller**: This controller is powered by the RP2040 chip and supports four motors with encoders[1].
2. **JGA25-370 Motors**: These are DC motors that can be used with the Motor 2040.
3. **Power Supply**: Ensure you have a suitable power supply for your motors and controller.
4. **JST-SH Cables**: These are used to connect the motors to the controller.
5. **Microcontroller Programming Environment**: You can use C++, MicroPython, or CircuitPython[2]. We used CircuitPython as it had support for turning the Motor Controller into a receiver.

Steps to Create Your Project:

1. **Connect the Motors**:
    - Attach the JGA25-370 motors to the Motor 2040 using the JST-SH cables. Make sure each motor is connected to one of the four motor channels on the controller.
2. **Power the Controller**:
    - Connect your power supply to the Motor 2040. The controller supports a wide voltage range (2.7V to 10V) for motors and logic[2].
3. **Set Up the Programming Environment**:
    - Install the necessary libraries for your chosen programming language. For example, if you're using MicroPython, you can find the Motor 2040 library and examples on the Pimoroni GitHub page[2].
4. **Write Your Code**:
    - Start by writing simple code to control the motors. For instance, you can write a script to make the motors move forward, backward, and stop. Here's a basic example in MicroPython:

```
from motor import Motor2040
   from time import sleep

   motor = Motor2040()

   # Move motor 1 forward
   motor.motor1.forward(0.5)
   sleep(2)

   # Stop motor 1
   motor.motor1.stop()
   sleep(1)

   # Move motor 1 backward
   motor.motor1.backward(0.5)
   sleep(2)

   # Stop motor 1
   motor.motor1.stop()
```

References

[1] Motor 2040 - Quad Motor Controller - Pimoroni

Understanding PID Control

PID control combines three components:

1. **Proportional (P)**: This term produces an output value that is proportional to the current error value. It helps reduce the overall error.
2. **Integral (I)**: This term is concerned with the accumulation of past errors. It helps eliminate the residual steady-state error that occurs with a pure proportional controller.
3. **Derivative (D)**: This term predicts future error based on its rate of change. It helps reduce the overshoot and improve the stability of the system.

Steps to Implement PID Control

1. **Define the PID Equation**: The PID control equation is: $ u(t) = K*p e(t)* + *K*i \int e(t) dt + K_d \frac{de(t)}{dt} $ where:

- (u(t)) is the control output.
- (e(t)) is the error (setpoint - measured value).
- (K*p), (K*i), and (K_d) are the proportional, integral, and derivative gains, respectively.

2. **Initialize Variables**: Set up variables for the PID gains, error terms, and control output.
3. **Calculate Error**: Compute the error between the desired setpoint and the actual measured value.
4. **Compute PID Output**: Calculate the proportional, integral, and derivative terms and sum them to get the control output.
5. **Apply Control Output**: Use the control output to adjust the motor speed or position.

Example Code in Python

Here's a basic example of a PID controller in Python:

```
class PID:
    def __init__(self, Kp, Ki, Kd):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.prev_error = 0
        self.integral = 0

    def compute(self, setpoint, measured_value):
        error = setpoint - measured_value
        self.integral += error
        derivative = error - self.prev_error
        output = self.Kp * error + self.Ki * self.integral + self.Kd *
derivative
        self.prev_error = error
        return output

# Example usage
pid = PID(Kp=1.0, Ki=0.1, Kd=0.05)
setpoint = 100  # Desired value
measured_value = 90  # Current value

control_output = pid.compute(setpoint, measured_value)
print(f"Control Output: {control_output}")
```

Tuning PID Parameters

Tuning the (K*p*), (*K*i), and (K_d) parameters is crucial for optimal performance. You can start with the Ziegler-Nichols method or trial and error to find the best values for your system[1][2].

Feel free to ask if you need more details or have specific questions about your project!

References

[1] PID controller basics & tutorial: PID implementation in Arduino

[2] PID controller implementation using Arduino - Microcontrollers Lab

Steps to Implement PID Control in Your Project

To implement PID control in your project with the Motor 2040 and JGA25-370 motors, you'll need to integrate the PID algorithm into your motor control code. Here's a step-by-step guide to help you:

1. **Set Up the PID Controller**:
    - Define the PID controller class as shown in the previous example.
2. **Initialize the PID Controller**:
    - Create an instance of the PID controller with appropriate gains.
3. **Read Encoder Values**:
    - Use the encoders on the Motor 2040 to get the current position or speed of the motors.
4. **Compute the PID Output**:
    - Calculate the control output using the PID controller based on the desired setpoint and the measured value from the encoders.
5. **Apply the Control Output to the Motors**:
    - Use the control output to adjust the motor speed or position.

Example Code

Here's an example of how you can integrate PID control into your project using MicroPython:

```
from motor import Motor2040
from time import sleep

class PID:
    def __init__(self, Kp, Ki, Kd):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.prev_error = 0
        self.integral = 0

    def compute(self, setpoint, measured_value):
        error = setpoint - measured_value
        self.integral += error
        derivative = error - self.prev_error
        output = self.Kp * error + self.Ki * self.integral + self.Kd *
derivative
        self.prev_error = error
        return output

# Initialize Motor 2040 and PID controller
motor = Motor2040()
pid = PID(Kp=1.0, Ki=0.1, Kd=0.05)

setpoint = 100  # Desired position or speed

while True:
    # Read the current position or speed from the encoder
    measured_value = motor.motor1.get_encoder_value()

    # Compute the PID control output
    control_output = pid.compute(setpoint, measured_value)

    # Apply the control output to the motor
    motor.motor1.set_speed(control_output)

    # Small delay to prevent overwhelming the controller
    sleep(0.01)
```

Tuning the PID Parameters

- **Proportional Gain (Kp)**: Start with a small value and increase it until you get a quick response without too much overshoot.

- **Integral Gain (Ki)**: Adjust this to eliminate steady-state error. Be cautious, as too high a value can cause instability.
- **Derivative Gain (Kd)**: Use this to reduce overshoot and improve stability. Start with a small value and increase it gradually.

Two Motor Controllers 3 Motors

Using two motor controllers to control six motors can be a bit more complex, but it's definitely doable! Here's a step-by-step guide to help you set up your project:

Materials Needed:

1. **Two Motor 2040 Controllers**: Each controller can handle up to four motors.
2. **Six JGA25-370 Motors**: These are the motors you'll be controlling.
3. **Power Supply**: Ensure you have a suitable power supply for your motors and controllers.
4. **JST-SH Cables**: For connecting the motors to the controllers.
5. **Microcontroller Programming Environment**: You can use C++, MicroPython, or CircuitPython.

Steps to Set Up Your Project:

1. **Connect the Motors to the Controllers**:
    - Divide the six motors between the two Motor 2040 controllers. For example, connect three motors to each controller using the JST-SH cables.
2. **Power the Controllers**:
    - Connect your power supply to both Motor 2040 controllers. Ensure the power supply can handle the combined current draw of all six motors.
3. **Set Up the Programming Environment**:
    - Install the necessary libraries for your chosen programming language. For example, if you're using MicroPython, you can find the Motor 2040 library and examples on the Pimoroni GitHub page.
4. **Write Your Code**:

- Write code to control the motors on both controllers. You'll need to initialize both controllers and manage the motors accordingly. Here's a basic example in MicroPython:

```
from motor import Motor2040
from time import sleep

# Initialize both Motor 2040 controllers
motor_controller_1 = Motor2040()
motor_controller_2 = Motor2040()

# Function to control motors on both controllers
def control_motors():
    # Move motors on controller 1
    motor_controller_1.motor1.forward(0.5)
    motor_controller_1.motor2.forward(0.5)
    motor_controller_1.motor3.forward(0.5)

    # Move motors on controller 2
    motor_controller_2.motor1.forward(0.5)
    motor_controller_2.motor2.forward(0.5)
    motor_controller_2.motor3.forward(0.5)

    sleep(2)

    # Stop all motors
    motor_controller_1.motor1.stop()
    motor_controller_1.motor2.stop()
    motor_controller_1.motor3.stop()
    motor_controller_2.motor1.stop()
    motor_controller_2.motor2.stop()
    motor_controller_2.motor3.stop()

# Run the control function
control_motors()
```

5. **Synchronize the Controllers**:
    - Ensure that both controllers are synchronized in terms of their control signals. You may need to use a common clock or communication protocol to achieve this.

Additional Tips:

- **Communication Between Controllers**: If you need the controllers to communicate, consider using I2C, SPI, or UART protocols.
- **Power Management**: Ensure your power supply can handle the total current draw of all six motors to avoid any power issues.
- **PID Control**: If you need precise control, you can implement PID control as discussed earlier and apply it to each motor individually.

Connect two Motor 2040 Controllers to Raspberry Pi

To connect a Raspberry Pi to two Motor 2040 controllers for controlling six motors, you'll need to set up communication between the Raspberry Pi and the motor controllers. Here's a step-by-step guide:

Materials Needed:

1. **Raspberry Pi**: Any model with GPIO pins (e.g., Raspberry Pi 4, Raspberry Pi 3, etc.).
2. **Two Motor 2040 Controllers**: Each controller can handle up to four motors.
3. **Six JGA25-370 Motors**: These are the motors you'll be controlling.
4. **JST-SH Cables**: For connecting the motors to the controllers.
5. **Power Supply**: Ensure you have a suitable power supply for your motors and controllers.
6. **I2C or UART Cables**: For communication between the Raspberry Pi and the motor controllers.

Steps to Connect and Control the Motor Controllers

1. **Wiring the Motor Controllers**:
    - Connect the motors to the Motor 2040 controllers using the JST-SH cables.
    - Connect the power supply to both Motor 2040 controllers.
2. **Connecting the Raspberry Pi to the Motor Controllers**:
    - **I2C Communication**:
        - Connect the SDA and SCL pins of the Raspberry Pi to the SDA and SCL pins of both Motor 2040 controllers.

        - Connect the ground (GND) of the Raspberry Pi to the ground (GND) of both Motor 2040 controllers.
    - **UART Communication** (alternative):
        - Connect the TX pin of the Raspberry Pi to the RX pin of the first Motor 2040 controller and the RX pin of the Raspberry Pi to the TX pin of the first Motor 2040 controller.
        - Repeat the connections for the second Motor 2040 controller.
3. **Setting Up the Programming Environment**:
    - Install the necessary libraries for I2C or UART communication on the Raspberry Pi. For example, you can use the smbus library for I2C communication in Python.
4. **Writing the Control Code**:
    - Write a Python script to control the motors via the Motor 2040 controllers. Here's an example using I2C communication for PWM control

```
# PWM control with two motor controllers
import smbus
from time import sleep

# Initialize I2C bus
bus = smbus.SMBus(1)  # 1 indicates /dev/i2c-1

# Motor 2040 I2C addresses
MOTOR_CONTROLLER_1_ADDR = 0x10
MOTOR_CONTROLLER_2_ADDR = 0x11

# Function to set motor speed
def set_motor_speed(controller_addr, motor_num, speed):
    bus.write_byte_data(controller_addr, motor_num, speed)

# Example usage
set_motor_speed(MOTOR_CONTROLLER_1_ADDR, 1, 128)  # Set motor 1 on
controller 1 to half speed
set_motor_speed(MOTOR_CONTROLLER_2_ADDR, 1, 128)  # Set motor 1 on
controller 2 to half speed

sleep(2)

# Stop motors
set_motor_speed(MOTOR_CONTROLLER_1_ADDR, 1, 0)
set_motor_speed(MOTOR_CONTROLLER_2_ADDR, 1, 0)
```

Synchronizing the Controllers:

- Ensure that both controllers receive commands simultaneously. You can achieve this by sending commands to both controllers in quick succession within your control loop.

Additional Tips:

- **Power Management**: Ensure your power supply can handle the total current draw of all six motors.
- **Feedback Loop**: Implement a feedback loop using encoders to maintain precise control over the motors.

Implement a PID Feedback Loop with Two Motor Controllers

Implementing a feedback loop using encoders is a great way to achieve precise control over your motors. Here's a step-by-step guide to help you set up a closed-loop control system with your Raspberry Pi, Motor 2040 controllers, and JGA25-370 motors:

1. **Connect the Encoders**:
    - Ensure that the encoders on your JGA25-370 motors are properly connected to the Motor 2040 controllers. The Motor 2040 supports encoder inputs, which will provide real-time feedback on motor position and speed.
2. **Read Encoder Values**:
    - Use the Motor 2040 library to read the encoder values. These values will be used to calculate the error between the desired setpoint and the actual position or speed of the motors.
3. **Implement the PID Controller**:
    - Use a PID controller to calculate the control output based on the error. The PID controller will adjust the motor speed to minimize the error.
4. **Apply the Control Output**:
    - Use the control output from the PID controller to adjust the motor speed via the Motor 2040 controllers.

Example Code in Python

Here's an example of how you can implement a feedback loop using encoders and a PID controller in Python:

```
from motor import Motor2040
from time import sleep
class PID:
    def __init__(self, Kp, Ki, Kd):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.prev_error = 0
        self.integral = 0

    def compute(self, setpoint, measured_value):
        error = setpoint - measured_value
        self.integral += error
        derivative = error - self.prev_error
       output = self.Kp * error + self.Ki * self.integral + self.Kd * derivative
        self.prev_error = error
        return output
# Initialize Motor 2040 controllers
motor_controller_1 = Motor2040()
motor_controller_2 = Motor2040()

# Initialize PID controller
pid = PID(Kp=1.0, Ki=0.1, Kd=0.05)

setpoint = 100  # Desired position or speed

while True:
    # Read the current position or speed from the encoder
    measured_value_1 = motor_controller_1.motor1.get_encoder_value()
    measured_value_2 = motor_controller_2.motor1.get_encoder_value()

    # Compute the PID control output
    control_output_1 = pid.compute(setpoint, measured_value_1)
    control_output_2 = pid.compute(setpoint, measured_value_2)

    # Apply the control output to the motors
    motor_controller_1.motor1.set_speed(control_output_1)
    motor_controller_2.motor1.set_speed(control_output_2)

    # Small delay to prevent overwhelming the controller
    sleep(0.01)
```

Tuning the PID Parameters

- **Proportional Gain (Kp)**: Start with a small value and increase it until you get a quick response without too much overshoot.
- **Integral Gain (Ki)**: Adjust this to eliminate steady-state error. Be cautious, as too high a value can cause instability.
- **Derivative Gain (Kd)**: Use this to reduce overshoot and improve stability. Start with a small value and increase it gradually.

Additional Tips

- **Ensure Proper Encoder Placement**: Proper placement and alignment of encoders are crucial for accurate feedback[1].
- **Signal Integrity**: Ensure that the encoder signals are clean and free from noise to maintain precise control[2].

References

[1] How Does Encoder Feedback Improve Accuracy in Motion Control?

[2] How Does Encoder Feedback Interact with Variable Frequency Drives?

Motor 2040 MicroPython

How to load MicroPython

- https://github.com/pimoroni/pimoroni-pico pico-vx.x.x-pimoroni-micropython.uf2
- Installing MicroPython
- Getting Started with pico and Thonny
- MicroPython Examples
- Motors and Motor 2040 Motor Library (API)