**Electronic Speed Controllers :**

#### 1. Purpose

The Electronic Speed Controller (ESC) is utilized to regulate the speed and direction of thrusters in the ROV system. This document details the control mechanisms and software integration to ensure reliable and precise motor performance.

#### 2. Software Requirements

* **Libraries:** Servo.h

#### 3. Software Architecture

* **Initialization:** Initialize PWM output pins connected to ESC signal inputs.

Typical PWM range:

* **1000 μs:** Minimum speed (reverse if supported).
* **1500 μs:** Neutral (stationary).
* **2000 μs:** Maximum speed (forward).
* **Control Loop:** Continuously monitor and apply thrust values received via serial communication.
* **Safety Mechanisms:** Stop motors during signal loss.

#### 4. Control Methods

* **Throttle Control:** Map joystick or input commands to PWM signals.
* **Control Modes:**
* **Individual Thruster Control:** Each thruster can be controlled independently.
* **Directional Control:** PWM signals are combined to achieve specific movements (e.g., forward, backward, rotate).
* **5. Error Handling**
* Detect signal loss using a timeout mechanism.
* Reset all PWM signals to neutral (1500 microseconds) on communication failure.
* **Thrusters:**
  + Driven by brushless motors.
  + Controlled individually or in pairs for directional control.

**ESC and Thruster Integration**

* **ESC Setup:**
  + ESCs require calibration to match PWM ranges. This is usually done once at startup.
  + Each ESC is assigned to a specific thruster.
* **PWM Signal Mapping:**
  + The code calculates PWM values dynamically based on input commands and control algorithms (e.g., joystick commands, and PID output).

#### Pseudocode

Global Variables:

|  |
| --- |
| Define constant THRUSTER\_PWM\_NEUTRAL = 1500  Define constant THRUSTER\_PWM\_MIN = 1000  Define constant THRUSTER\_PWM\_MAX = 2000 |

Initialize PWM variables for all thrusters:

|  |
| --- |
| Thruster\_intLeftFrontPWM = THRUSTER\_PWM\_NEUTRAL  Thruster\_intLeftBackPWM = THRUSTER\_PWM\_NEUTRAL  Thruster\_intRightFrontPWM = THRUSTER\_PWM\_NEUTRAL  Thruster\_intRightBackPWM = THRUSTER\_PWM\_NEUTRAL  Thruster\_intUpFrontPWM = THRUSTER\_PWM\_NEUTRAL  Thruster\_intUpBackPWM = THRUSTER\_PWM\_NEUTRAL  Thruster\_intDownFrontPWM = THRUSTER\_PWM\_NEUTRAL  Thruster\_intDownBackPWM = THRUSTER\_PWM\_NEUTRAL |

**Function:** Thruster\_voidParseCommand

**Purpose:** Parse the incoming command string and update the PWM values for each thruster.

|  |
| --- |
| FUNCTION Thruster\_voidParseCommand(Copy\_strCommand)  FOR each thruster label ('A' to 'H'):  Extract PWM value using Thruster\_intGetPWMValue(Copy\_strCommand, thruster\_label)  Update the corresponding PWM variable:  IF label = 'A': Thruster\_intLeftFrontPWM = Extracted PWM value  IF label = 'B': Thruster\_intLeftBackPWM = Extracted PWM value  IF label = 'C': Thruster\_intRightFrontPWM = Extracted PWM value  IF label = 'D': Thruster\_intRightBackPWM = Extracted PWM value  IF label = 'E': Thruster\_intUpFrontPWM = Extracted PWM value  IF label = 'F': Thruster\_intUpBackPWM = Extracted PWM value  IF label = 'G': Thruster\_intDownFrontPWM = Extracted PWM value  IF label = 'H': Thruster\_intDownBackPWM = Extracted PWM value  END FOR  END FUNCTION |

**Function:** Thruster\_intGetPWMValue

**Purpose:** Extract the PWM value for a specific thruster from the command string.

|  |
| --- |
| FUNCTION Thruster\_intGetPWMValue(Copy\_strCommand, Copy\_charLabel)  StartIndex = Find position of Copy\_charLabel in Copy\_strCommand  EndIndex = Find next space or end of string after StartIndex  PWMValue = Convert substring between StartIndex and EndIndex to integer  RETURN PWMValue  END FUNCTION |

**Function: Thruster\_voidApplyPWM**

**Purpose:** Apply the calculated PWM values to the ESCs controlling the thrusters.

|  |
| --- |
| FUNCTION Thruster\_voidApplyPWM()  FOR each thruster servo:  IF servo = Thruster\_SERLeftFront:  Send Thruster\_intLeftFrontPWM to ESC  IF servo = Thruster\_SERLeftBack:  Send Thruster\_intLeftBackPWM to ESC  IF servo = Thruster\_SERRightFront:  Send Thruster\_intRightFrontPWM to ESC  IF servo = Thruster\_SERRightBack:  Send Thruster\_intRightBackPWM to ESC  IF servo = Thruster\_SERUpFront:  Send Thruster\_intUpFrontPWM to ESC  IF servo = Thruster\_SERUpBack:  Send Thruster\_intUpBackPWM to ESC  IF servo = Thruster\_SERDownFront:  Send Thruster\_intDownFrontPWM to ESC  IF servo = Thruster\_SERDownBack:  Send Thruster\_intDownBackPWM to ESC  END FOR  END FUNCTION |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### **Servo Motor Control for Camera**

#### 1. Purpose

To provide precise control over the camera’s orientation using servo motors.

#### 2. Software Requirements

* **Libraries:** Servo.h

**Signal Input:** PWM (0-180 degrees).

#### 3. Software Architecture

* **Initialization:** Configure servo motor pins for PWM output.
* **Control Loop:** Process joystick input to determine camera orientation.
* **Safety Mechanisms:** Reset servo to default position on system reset.

#### 4. Position Control Methods

* **Mapping Input:** Convert joystick input into angular positions.
  + Map the raw joystick values (0–1023) to servo angles (0–180).
* **Smooth Transition:** Gradually change position to avoid mechanical stress.

#### 5. Error Handling

* Implement boundary checks to ensure positions remain within servo’s physical limits.
* Reset to neutral position on system reset.

#### 6. Pseudocode

|  |
| --- |
| FUNCTION setup()  Attach servoX to pin 6  Attach servoY to pin 7  Set buttonPin as INPUT with pull-up  Set initial servo positions to 90° (default)  Start Serial communication  Print initialization message  END FUNCTION |

|  |
| --- |
| FUNCTION loop()  Read analog joystick values:  joystickValX = analogRead(joystickX)  joystickValY = analogRead(joystickY)  Map joystick values (0–1023) to servo angles (0–180):  angleX = map(joystickValX, 0, 1023, minAngle, maxAngle)  angleY = map(joystickValY, 0, 1023, minAngle, maxAngle)  Write angles to servos:  servoX.write(angleX)  servoY.write(angleY)  Print angles to Serial Monitor  IF reset button is pressed:  Call resetPosition()  Delay 100ms  END FUNCTION |