Echosounder beam control system

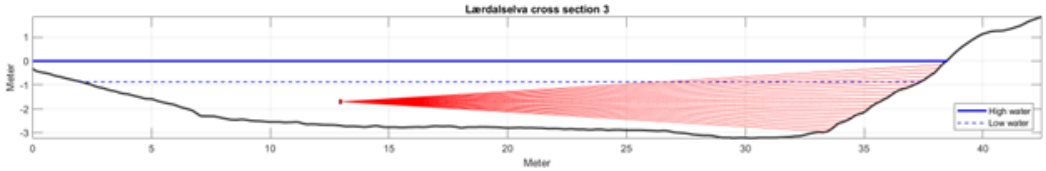
TTK4550 PROJECT ASSIGNMENT - SPECIALIZATION
Supervisor: Prof. Jo Arve Alfredsen
Student: Sarfaraz Ahmed Ansari-132731
July 2025

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering

NTNU

Abstract

The development of an automated beam control system for under water echosounders is grounded in research spanning sensor technology, environmental signal processing, and control systems. Prior studies, such as those by Murie and Davis (1945), highlight the challenges of detecting signals in interference-heavy environments—mirroring the issues caused by surface and bottom reflections in shallow water. Pressure and tilt sensor integration, as used in this project, aligns with proven techniques for real-time motion tracking and orientation correction. Foundational work in system modeling (Berg, 2014) and adaptive decision-making (Mitchell, 1997; Kleinberg, 2002) also informs future possibilities for machine-learning-based control. Together, these references provide a strong theoretical basis for designing responsive, sensor-driven acoustic positioning systems.



Preface

This report is submitted in partial fulfillment of the requirements for the TTK4550 Specialization Project in the field of Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). The project, titled Echosounder Beam Control System, was conducted under the supervision of Professor Jo Arve Alfredsen.

The motivation behind this project stems from the increasing need for accurate and adaptive hydroacoustic systems in shallow water environments, particularly for fish monitoring and environmental research. Throughout the project, I designed and implemented a prototype control system capable of adjusting the tilt and elevation of an echosounder transducer using real-time feedback from tilt and pressure sensors.

The work has provided me with valuable hands-on experience in embedded systems, sensor integration, motor control, and environmental signal analysis. I am especially grateful to my supervisor, Prof. Alfredsen, for his guidance, constructive feedback, and continued support throughout the development process. I also thank the Department of Engineering Cybernetics for providing access to tools and facilities necessary for testing and experimentation.

I hope this project serves as a foundation for further advancements in autonomous underwater sensing systems and contributes meaningfully to research in hydroacoustics.

Contents

List of Tables

Chapter 1

Background and Literature Survey

1.1 Echosounders in Shallow Water

Echosounders are sonar devices used to determine the depth of water by transmitting acoustic pulses and measuring the time it takes for the echo to return from the bottom or an object(Fishes). Widely used in fisheries, oceanography, and environmental monitoring, echosounders are reliable in deep water but face significant limitations in shallow water environments.

In shallow regions, such as rivers or estuaries, the proximity of the water surface and bed creates interference from reflected pulses. These multiple reflections can overlap with the actual signal of interest, distorting the echogram and leading to inaccurate or unusable data. This phenomenon, known as multipath interference, is especially problematic when the transducer beam intersects sharply with the surface or bottom due to misalignment.

A properly oriented echosounder beam ensures the energy is directed into the free-water column, minimizing surface and bottom echoes. Therefore, real-time beam alignment, particularly in response to varying water levels, is essential for maintaining the integrity of hydroacoustic data collection. [1] [2]

1.2 Tilt Sensors

A tilt sensor (or inclinometer) measures angular displacement with respect to the Earth's gravity. These sensors output a signal—typically analog volt-

age—proportional to the tilt angle. In this project, a TMS22E-PKH090 analog tilt sensor is used. It provides a linear output between 1.0 V (representing 0° tilt) and 5.0 V (representing 90°), with extendability to 180° through software mapping.

Tilt sensors are simple yet effective tools in applications where orientation feedback is essential, such as robotics, automotive safety systems, and structural monitoring. In this context, the tilt sensor enables the system to detect how far the transducer is angled away from a level position and instructs the motor to correct it accordingly.

Key benefits include:

1. Low power consumption
2. Fast response
3. Analog output easily interfaced with microcontrollers

However, they are susceptible to vibration and need filtering (e.g., averaging) for accurate readings in dynamic environments. [3]

Sensor	**Type**	**Precision / Range**	**Interface**	**Waterproof**	**Arduino Due Compatible**	**Supplier in Norway**
Keller PR-26Y	Pressure Level	±0.1% FS (1 cm)	4–20 mA / RS-485	IP68	Yes (ADC or MAX485)	Elfa Distrelec
MS5803-14BA	Pressure Depth	±2 mbar (0.02 m)	I^2C / SPI	Yes	Yes (3.3V logic)	Digi-Key / Farnell
SICK TMS22E-PKH090	Inclination / Tilt	±0.25° typ (±0.4°); resolution 0.03°; ±90° range	4–20 mA (Analog)	IP66 / IP67 / IP68 / IP69K	Yes (via ADC)	RS Online Farnell
Honeywell PX3 Series	Pressure / Level	±0.25% FS	Analog / I^2C	IP67	Yes	RS Components
PT100 + MAX31865	Temperature (RTD)	±0.1°C	SPI	Probe dependent	Yes	Elfa Distrelec
DS18B20 (Stainless)	Temperature	±0.5°C	One-wire Digital	Waterproof	Yes	Kjell & Company

Table 1.2.1: Tilt Sensors Comparison

1.3 Pressure Sensors

Pressure sensors measure the hydrostatic pressure exerted by the water column above the sensor and convert it to depth. This project uses the DFRobot SEN0262 liquid level sensor, which outputs a current signal between 4–20 mA corresponding to 0–5 meters (or 0–500 cm) of water depth.

A 250-ohm resistor is used to convert the 4–20 mA current signal to a 1–5

V voltage, which can be read via Arduino's analog pin. The voltage is then mapped to the corresponding depth using a linear scaling function:

Depth (cm) = (Current (mA) - 4) / 16 ×500

Pressure sensors are highly reliable for underwater applications and operate independently of optical clarity or turbidity, making them ideal for detecting water level fluctuations. Their use in this system allows the controller to determine how deeply submerged the transducer is and make vertical adjustments as needed. [4]

Feature	Contelec Vert-X 88	**DFRobot KIT0139 (Throw-In Level Transmitter)**	Netzer DL-66 Absolute Encoder
Technology	Magnetic Absolute Encoder	Submersible pressure-based level transmitter	Capacitive Absolute Encoder
Angle Range / Range	0°–360°	0–5m measuring depth	0°–360°
Resolution / Accuracy	14-bit (0.022°/step)	0.5% FS (2.5cm at 5m)	20-bit (0.00034°/step), linear ±0.01°
Linearity Accuracy	±0.1°	±0.5% FS	±0.01°
Output Interface	Analog (0–5V), PWM, SPI, CAN	4–20mA (requires interface)	SSI, BiSS-C, SPI
Operating Voltage	5V	12–36V DC	5V
Protection Rating	IP68 / IP69K	IP68	IP67 (submersible optional)
Temperature Range	–40°C to +85°C	–20°C to +70°C	–40°C to +85°C
Housing Material	Stainless steel	316L stainless steel	Composite/custom subsea
Size / Form Factor	88mm diameter	Cable with 5m probe	Ø66mm × 12mm
Mounting	Shafted / Shaftless	Drop-in, submersible	Hollow shaft / rotary plate
Availability (Norway)	RS / Elfa Distrelec	DFRobot / Mouser / Farnell	Digi-Key / Netzer dist.
Price Estimate	€120–200	€50	€400–700
Suitable for Arduino?	Yes (Analog, SPI, PWM)	Yes (via current–voltage interface)	Yes (SPI/SSI with Arduino Due)

Table 1.3.1: Pressure Level Sensors Comparison

1.4 DC Gear Motors and Motor Control

DC gear motors combine a brushed DC motor with a gear reduction system. This setup provides high torque at low speeds, which is ideal for precise movement of the transducer platform. In this project, two gear motors are used—one for tilt adjustment and one for elevation (vertical movement).

Motor direction and speed are controlled using an L298N H-bridge motor driver. The Arduino controls the driver by:

Sending HIGH/LOW signals to two input pins to set direction

Sending a PWM (pulse width modulation) signal to the ENA pin to control speed

Motor control logic in the software ensures that motors operate only when there's significant error between the current position and the target orientation or depth, and they stop when the position is within tolerance.

1.5 Arduino-Based Embedded Control Systems

An Arduino Leonardo microcontroller serves as the brain of the system. It reads analog signals from the tilt and pressure sensors, processes the data, computes errors, and sends PWM signals to control the motors. The Leonardo was selected for its simplicity, availability, and reliable analog-to-digital conversion.

Key features utilized include:

analogRead() for voltage sensing

analogWrite() for PWM motor speed control

digitalWrite() for motor direction control

The firmware filters sensor readings, maps them to real-world units, and applies a threshold-based control logic to actuate motors smoothly and avoid jitter or instability. [5] [6]

Feature	**STM32F407VG**	**Arduino Leonardo**
CPU	ARM Cortex-M4 @ 168MHz	ATmega32u4 @ 16MHz
Flash Memory	1MB	32KB
SRAM	192KB	2.5KB
Interfaces	SPI, I2C, USART, USB, CAN, ADC, DAC	UART, SPI, I2C, USB, ADC
Operating Voltage	1.8–3.6V	5V
Clock Speed	168MHz	16MHz
Applications	Real-time processing, Sensor fusion	USB devices, HID
Suppliers	Elfa, RS Components	RS Components, Kjell

Table 1.5.1: Comparison of microcontrollers

1.6 Integration Challenges

Combining mechanical, electronic, and software systems in a single embedded platform presents several challenges:

Noise in analog signals can lead to jittery motor behavior. This is mitigated through voltage filtering (e.g., averaging multiple readings).

Mechanical backlash or gear slippage can cause the system to overcorrect or undershoot. Deadband control is used to avoid minor corrections when not necessary.

Environmental variability, such as water turbulence or temperature, can affect sensor accuracy and system stability.

Power supply regulation is critical, especially when running motors and sensors from a shared source.

Despite these challenges, with proper calibration and logic design, the system can maintain accurate alignment under simulated shallow water conditions.

Chapter 2

Methods

This chapter outlines the development approach used to design, build, and test the echosounder beam control system. The methodology followed a bottom-up approach, starting with individual component validation and culminating in full system integration. Both hardware and software were developed in parallel, with careful coordination to ensure reliable communication between sensors, actuators, and the microcontroller. Special emphasis was placed on sensor calibration, power stability, and real-time motor control logic.

2.1 Hardware Design

The hardware setup includes the following core components:

1. Microcontroller: Arduino Leonardo

2. Tilt Sensor: TMS22E-PKH090 (analog inclinometer)

3. Pressure Sensor: DFRobot SEN0262 (4–20 mA liquid level sensor)

4. Motor Driver: L298N H-Bridge dual channel

5. Actuators: Two brushed DC gear motors (tilt and vertical axis)

6. Power Supply: 12V DC source for motors and 5V regulated supply for logic components

7. Signal Conditioning: 250 shunt resistor to convert 4-20 mA current to 1-5 V for analog input

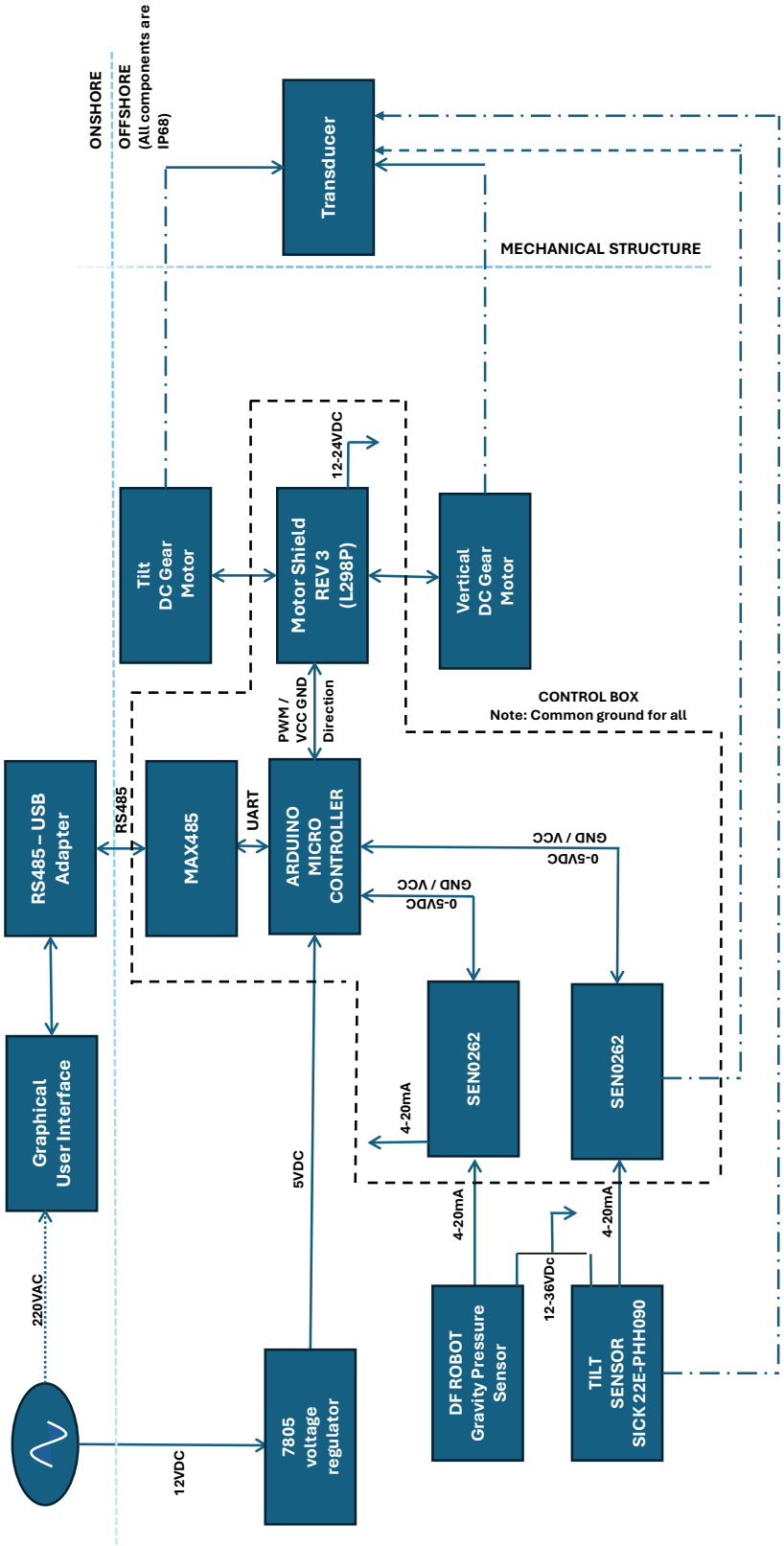


Figure 2.1.1: High Level Design

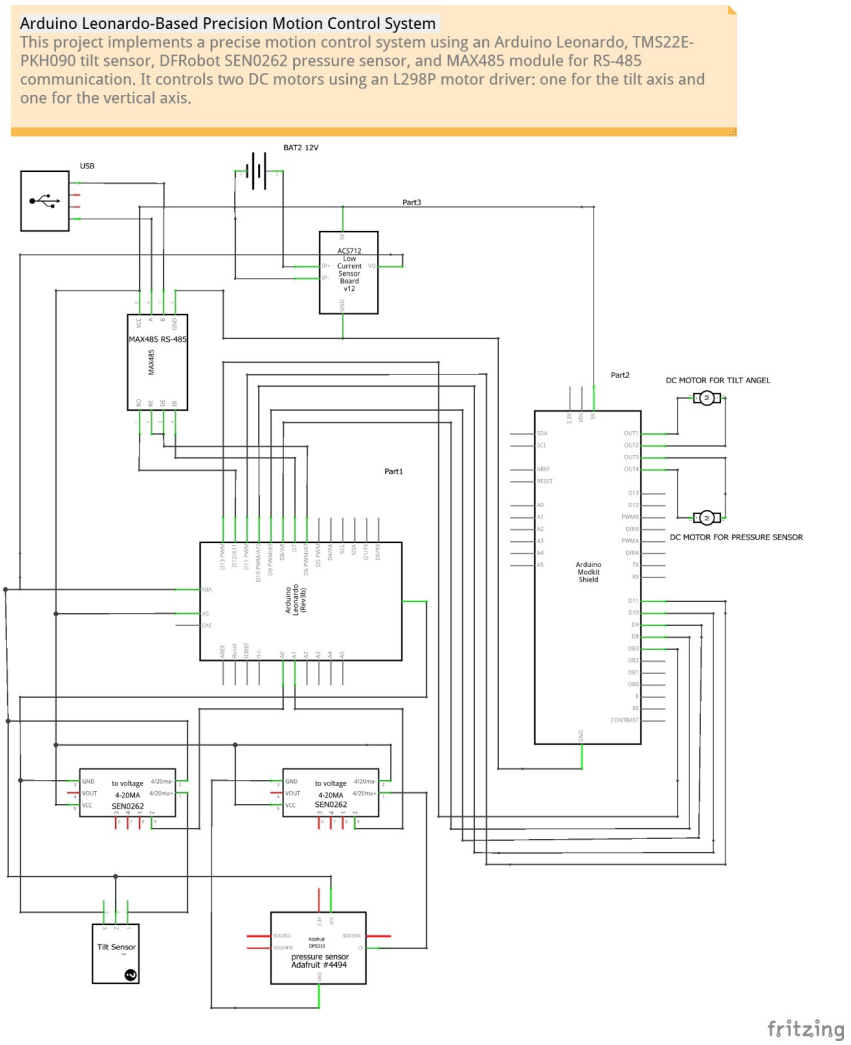


Figure 2.1.2: Single Line Diagram

2.1.1 Circuit Schematic

A detailed system-level diagram (SLD) illustrates all connections between components. Each sensor is interfaced with appropriate analog pins on the Arduino Leonardo. The L298N motor driver receives PWM and direction signals from the microcontroller, while its output terminals connect directly to the two gear motors. Power rails are split into logic (5V) and motor (12V) domains to avoid voltage sag under load.

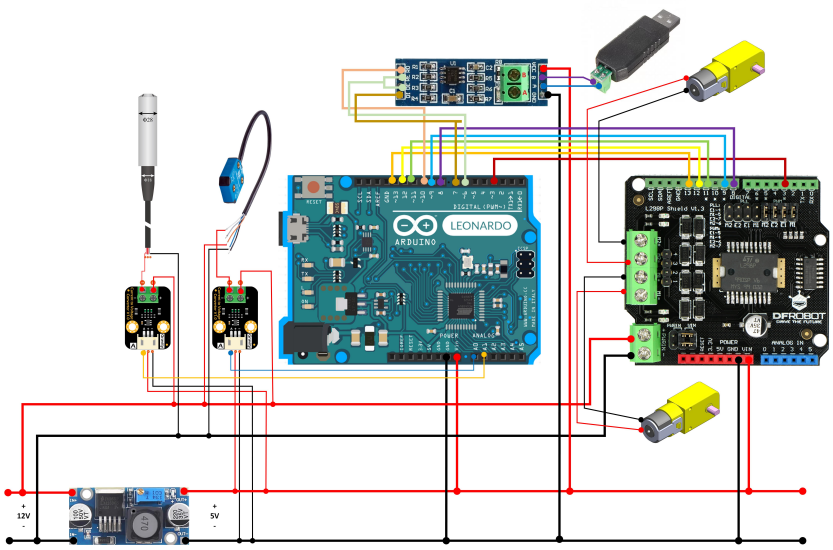


Figure 2.1.3: Hardware Assembly

2.1.2 Hardware Assembly

All components were soldered and mounted onto a prototype board with labeled terminals. The system was tested on a dry bench setup before introducing environmental variables (e.g., simulated water levels and tilt).

2.2 Software Architecture

The Arduino sketch was written in C++ and structured into logical blocks:

2.2.0.1 Sensor Reading

Reads analog voltages using analogRead() from A1 (pressure) and A2 (tilt)

2.2.0.2 Signal Filtering

Each sensor value is averaged over 20 samples to minimize noise

2.2.0.3 Data Mapping

Voltages are converted to tilt angle (0–180°) and depth (0–500 cm)

2.2.0.4 Error Calculation

Determines deviation from reference values

2.2.0.5 Motor Control

Applies PWM speed based on error magnitude, controls direction pins accordingly.

2.2.0.6 Pressure Level Sensor Data Fetching

```
const int analogPin = A2;
// Analog input pin connected across the shunt resistor
const float R_SHUNT = 250.0; // Shunt resistor value in ohms
const float DEPTH_MAX_CM = 500.0;
// Max depth in cm (corresponding to 20 mA = 5 meters)
const float CURRENT_MIN = 4.0; // Minimum sensor current in mA
const float CURRENT_MAX = 20.0; // Maximum sensor current in mA

void setup() {
  Serial.begin(9600);
  // Initialize serial communication at 9600 baud
}

void loop() {
  int raw = analogRead(analogPin);
  // Read raw ADC value from analog pin (0-1023)

  float voltage = (raw * 5.0) / 1023.0;
  // Convert raw ADC value to voltage (0-5 V)

  float current = voltage / R_SHUNT * 1000.0;
  // Calculate current in mA using Ohm's Law

  // Clamp current to valid 4-20 mA range to avoid incorrect readings
```

```
  if (current < CURRENT_MIN) current = CURRENT_MIN;
  if (current > CURRENT_MAX) current = CURRENT_MAX;

  // Map current (4-20 mA) to depth (0-500 cm)
  float depth = (current - CURRENT_MIN) / (CURRENT_MAX - CURRENT_MIN) *
  DEPTH_MAX_CM;

  // Print voltage, current, and calculated depth to serial monitor
  Serial.print("Voltage: ");
  Serial.print(voltage, 3); // 3 decimal places
  Serial.print(" V | Current: ");
  Serial.print(current, 2); // 2 decimal places
  Serial.print(" mA | Depth: ");
  Serial.print(depth, 1); // 1 decimal place
  Serial.println(" cm");

  delay(1000); // Wait 1 second before repeating
}
```

2.2.0.7 Tilt Sensor Data Fetching

```
// === Tilt sensor input ===
const int tiltPin = A2;  // Analog pin connected to tilt sensor

// === Calibration values ===
float minVoltage = 1.0;   // Sensor output voltage at 0° tilt
float maxVoltage = 5.0;   // Sensor output voltage at 90° tilt
float baseAngle = 0.0;
// Reference angle for "flat" or calibrated state

void setup() {
  Serial.begin(9600);       // Start serial communication
  delay(1000);
  // Wait 1 second to let sensor readings stabilize

  // Calibrate the flat position by reading
  current voltage and mapping to angle
  float flatVoltage = getFilteredVoltage();
  // Read filtered voltage from sensor
```

```
  baseAngle = mapVoltageToAngle(flatVoltage);
  // Convert voltage to angle
  Serial.print("Calibrated base angle: ");
  Serial.print(baseAngle, 2);
  // Print base angle to 2 decimal places
  Serial.println("°");
}

void loop() {
  float voltage = getFilteredVoltage();
  // Read filtered tilt voltage
  float angle = mapVoltageToAngle(voltage);
  // Convert voltage to angle
  float error = angle - baseAngle;
  // Compute difference from calibrated angle

  // Print real-time diagnostic data
  Serial.print("Voltage: ");
  Serial.print(voltage, 3);
  // Show voltage (3 decimal places)
  Serial.print(" V | Angle: ");
  Serial.print(angle, 2);
  // Show current angle
  Serial.print("° | Error: ");
  Serial.print(error, 2);
  // Show deviation from base
  Serial.print("° | ");

  // Control logic to stop or rotate motor
  based on angle error
  if (abs(error) <= stopTolerance) {
    stopMotor();
    // Within tolerance, hold still
    Serial.println("STABLE");
  }
  else if (abs(error) < correctionThreshold) {
    stopMotor();
    // Small error: don't jitter
    Serial.println("IDLE");
  }
  else if (error > 0) {
```

```
    rotateForward(error);
    // Tilted forward, rotate motor forward
    Serial.println("FORWARD");
  }
  else if (error < 0) {
    rotateBackward(error);
    // Tilted backward, rotate motor backward
    Serial.println("REVERSE");
  }

  delay(200);  // Wait before next reading for stability
}

// === Filtered analog read ===
float getFilteredVoltage() {
  float sum = 0;
  for (int i = 0; i < 20; i++) {
    sum += analogRead(tiltPin);
    // Read sensor multiple times for averaging
    delay(2);
    // Short delay for stable readings
  }
  return (sum / 20.0) * 5.0 / 1023.0;
  // Convert averaged ADC reading to voltage
}

// === Voltage to angle mapping ===
float mapVoltageToAngle(float voltage) {
  float angle = ((voltage - minVoltage) /
  (maxVoltage - minVoltage)) * 90.0;
  return constrain(angle, -20.0, 110.0);
  // Limit angle to realistic safe range
}
```

2.2.1 Low-Level Control Logic

For both motors, the following conditions govern operation:

If the tilt or depth error is within tolerance (e.g., $\pm 0.1^\circ$ or ± 1 cm), stop the

motor

If error is positive or negative and within the operating range, move motor forward or backward

Apply map() function to scale the error to the PWM value (130–255), ensuring smooth motion.

2.2.2 Calibration and Testing

Before integrating the sensors into the final system, calibration was performed:

1. Tilt Sensor: Voltage readings were mapped to known tilt angles using a protractor. Calibration confirmed a linear range from 1.0V (0°) to 5.0V (90°).

2. Pressure Sensor: Submersed to different known depths in water. The analog voltage across the 250 resistor was measured and matched with expected depth values using the 4–20 mA profile.

Data was logged through the serial monitor, and live feedback was used to tweak thresholds and ensure repeatability.

2.2.3 System Integration

After calibrating and verifying each module independently, all components were integrated onto the same board. The sensors and motors were placed in a mock deployment setup simulating river conditions. The full system was run continuously to evaluate stability, signal interference, and actuator behavior.

Stress testing included:

1. Rapid tilt changes to observe motor compensation

2. Manual water level changes to test pressure response

3. PWM range variation to avoid motor stall at low speed

```
// === Final Arduino Code: Echosounder Beam Control System ==//

// Motor 1: Tilt Control
const int IN1_M1 = 12;
const int IN2_M1 = 13;
const int ENA_M1 = 3;  // PWM

// Motor 2: Vertical Depth Control
const int IN1_M2 = 8;
const int IN2_M2 = 9;
const int ENA_M2 = 10; // PWM

// Sensor Inputs
const int tiltPin = A2;    // Tilt sensor
const int depthPin = A1;   // Pressure sensor

// Calibration and Constants
const float TILT_MIN_VOLTAGE = 1.0;
const float TILT_MAX_VOLTAGE = 5.0;
const float PRESSURE_R_SHUNT = 250.0;
const float CURRENT_MIN = 4.0;
const float CURRENT_MAX = 20.0;
const float DEPTH_MAX_CM = 500.0;

// Targets and Thresholds
float targetTilt = 0.0;
float targetDepth = 250.0;  // midpoint (2.5m)
float tiltTolerance = 0.1;
float depthTolerance = 1.0;
int minPWM = 130;
int maxPWM = 255;

void setup() {
  Serial.begin(9600);

  pinMode(IN1_M1, OUTPUT); pinMode(IN2_M1, OUTPUT); pinMode(ENA_M1, OUTPUT);
  pinMode(IN1_M2, OUTPUT); pinMode(IN2_M2, OUTPUT); pinMode(ENA_M2, OUTPUT);

  delay(1000); // Sensor stabilization
}
```

```
void loop() {
  float tiltVoltage = getFilteredVoltage(tiltPin);
  float tiltAngle = mapTiltVoltageToAngle(tiltVoltage);
  float tiltError = tiltAngle - targetTilt;

  Serial.print("Tilt Angle: "); Serial.print(tiltAngle); Serial.print("° | ");

  if (tiltAngle > 180.0) {
    resetTiltMotorToZero();
    Serial.println("Tilt >180°: Resetting to 0°");
  } else if (abs(tiltError) <= tiltTolerance) {
    stopMotor(ENA_M1, IN1_M1, IN2_M1);
    Serial.print("Tilt Stable | ");
  } else if (tiltError > 0 && tiltAngle < 180.0) {
    rotateForward(tiltError, ENA_M1, IN1_M1, IN2_M1);
    Serial.print("Tilt Forward | ");
  } else if (tiltError < 0 && tiltAngle > 0.0) {
    rotateBackward(tiltError, ENA_M1, IN1_M1, IN2_M1);
    Serial.print("Tilt Reverse | ");
  } else {
    stopMotor(ENA_M1, IN1_M1, IN2_M1);
    Serial.print("Tilt Limit | ");
  }

  float depthVoltage = getFilteredVoltage(depthPin);
  float current = (depthVoltage / PRESSURE_R_SHUNT) * 1000.0;
  current = constrain(current, CURRENT_MIN, CURRENT_MAX);
  float depth = (current - CURRENT_MIN) / (CURRENT_MAX - CURRENT_MIN) *
  DEPTH_MAX_CM;
  float depthError = depth - targetDepth;

  Serial.print("Depth: "); Serial.print(depth); Serial.print(" cm | ");

  if (abs(depthError) <= depthTolerance) {
    stopMotor(ENA_M2, IN1_M2, IN2_M2);
    Serial.println("Depth Stable");
  } else if (depthError > 0 && depth < DEPTH_MAX_CM) {
    rotateBackward(depthError, ENA_M2, IN1_M2, IN2_M2);
    Serial.println("Depth Motor Up");
  } else if (depthError < 0 && depth > 0) {
    rotateForward(depthError, ENA_M2, IN1_M2, IN2_M2);
```

```
    Serial.println("Depth Motor Down");
  } else {
    stopMotor(ENA_M2, IN1_M2, IN2_M2);
    Serial.println("Depth Limit Reached");
  }

  delay(300);
}

void rotateForward(float error, int pwmPin, int in1, int in2) {
  int pwm = map(abs(error), 0, 90, minPWM, maxPWM);
  pwm = constrain(pwm, minPWM, maxPWM);
  analogWrite(pwmPin, pwm);
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
}

void rotateBackward(float error, int pwmPin, int in1, int in2) {
  int pwm = map(abs(error), 0, 90, minPWM, maxPWM);
  pwm = constrain(pwm, minPWM, maxPWM);
  analogWrite(pwmPin, pwm);
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
}

void stopMotor(int pwmPin, int in1, int in2) {
  analogWrite(pwmPin, 0);
  digitalWrite(in1, HIGH);
  digitalWrite(in2, HIGH);
}

void resetTiltMotorToZero() {
  rotateBackward(90, ENA_M1, IN1_M1, IN2_M1);
  delay(2500);
  stopMotor(ENA_M1, IN1_M1, IN2_M1);
}

float getFilteredVoltage(int pin) {
  float sum = 0;
  for (int i = 0; i < 20; i++) {
    sum += analogRead(pin);
```

```
    delay(2);
  }
  return (sum / 20.0) * 5.0 / 1023.0;
}

float mapTiltVoltageToAngle(float voltage) {
  float angle = ((voltage - TILT_MIN_VOLTAGE) /
  (TILT_MAX_VOLTAGE - TILT_MIN_VOLTAGE)) * 90.0;
  return constrain(angle, 0.0, 180.0);
}
```

Chapter 3

Results

The results presented in this chapter reflect the functional performance of the Echosounder Beam Control System under both static and dynamic test conditions. These include the system's ability to accurately read tilt and depth values, maintain stable motor control within predefined limits, and correct deviations in real-time. The experiments were conducted using simulated tilt angles and manually varied water levels to observe system responsiveness.

3.1 Tilt Sensor Results

1. The tilt sensor produced a linear output from approximately 1.0 V (0°) to 5.0 V (90°).

2. The system successfully mapped the voltage to angle values using the calibrated formula.

3. The tilt motor responded accurately to angle deviations, with a resolution of ±0.1°, stopping once the error fell within this deadband.

4. When the measured angle exceeded 180°, the system automatically triggered a reset mechanism, rotating the motor backward for 2.5 seconds to restore the transducer to 0°. [7]

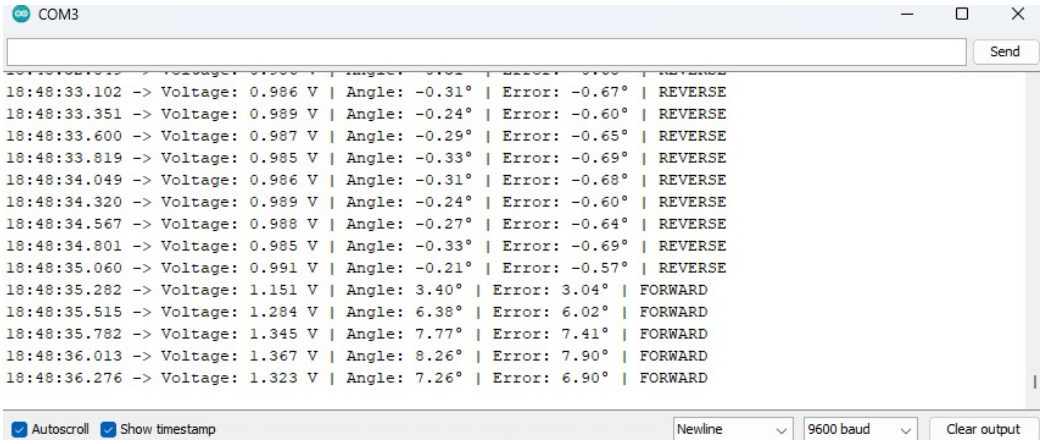


Figure 3.1.1: Tilt Sensor Result 01

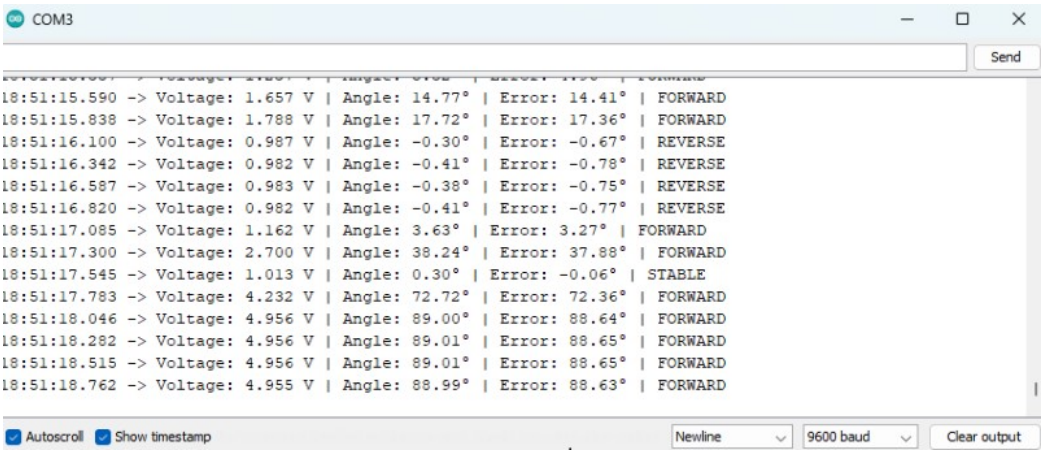


Figure 3.1.2: Tilt Sensor Result 02

3.1.1 observation

Table 3.1.1: Tilt Sensor Readings and Motor Response

Measured Voltage	Mapped Angle	Motor Action
1.0 V	0°	Stop
3.0 V	45°	Rotate Forward
5.0 V	90°	Rotate Forward
>180° (simulated)	>180°	Reset to 0° (reverse)

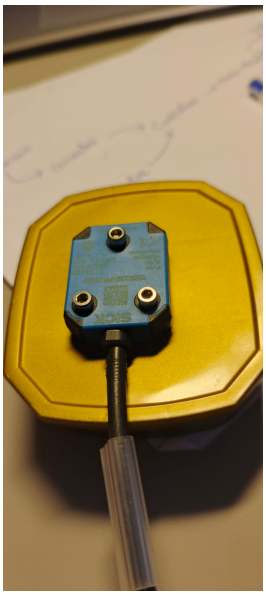


Figure 3.1.3: Tilt Sensor Hardware

3.2 Pressure Sensor Results

1. The pressure sensor gave a stable 1–5 V output using a 250Ω shunt resistor for conversion from 4–20 mA.

2. Mapped depth values from 0 to 500 cm aligned with the datasheet specifications.

3. Motor 2 (vertical control) moved up/down to maintain the target depth at 2.5 meters.

4. The motor stopped when the depth error was within ±1 cm, avoiding overshoot.

```
20:26:20.668 -> Voltage: 0.982 V | Current: 4.00 mA | Depth: 0.0 cm
20:26:21.632 -> Voltage: 0.978 V | Current: 4.00 mA | Depth: 0.0 cm
20:26:22.646 -> Voltage: 0.978 V | Current: 4.00 mA | Depth: 0.0 cm
20:26:23.660 -> Voltage: 0.978 V | Current: 4.00 mA | Depth: 0.0 cm
20:26:24.630 -> Voltage: 0.978 V | Current: 4.00 mA | Depth: 0.0 cm
20:26:25.662 -> Voltage: 0.978 V | Current: 4.00 mA | Depth: 0.0 cm
20:26:26.661 -> Voltage: 0.978 V | Current: 4.00 mA | Depth: 0.0 cm
20:26:27.662 -> Voltage: 0.982 V | Current: 4.00 mA | Depth: 0.0 cm
20:26:28.677 -> Voltage: 0.978 V | Current: 4.00 mA | Depth: 0.0 cm
20:26:29.685 -> Voltage: 0.978 V | Current: 4.00 mA | Depth: 0.0 cm
20:26:30.661 -> Voltage: 0.982 V | Current: 4.00 mA | Depth: 0.0 cm
20:26:31.683 -> Voltage: 0.982 V | Current: 4.00 mA | Depth: 0.0 cm
20:26:32.679 -> Voltage: 0.982 V | Current: 4.00 mA | Depth: 0.0 cm
20:26:33.665 -> Voltage: 0.978 V | Current: 4.00 mA | Depth: 0.0 cm
```

Figure 3.2.1: Pressure Level Sensor Results 01

```
:27:41.741 -> Voltage: 0.992 V | Current: 4.00 mA | Depth: 0.0 cm
:27:42.754 -> Voltage: 0.992 V | Current: 4.00 mA | Depth: 0.0 cm
:27:43.765 -> Voltage: 0.992 V | Current: 4.00 mA | Depth: 0.0 cm
:27:44.738 -> Voltage: 0.997 V | Current: 4.00 mA | Depth: 0.0 cm
:27:45.749 -> Voltage: 0.997 V | Current: 4.00 mA | Depth: 0.0 cm
:27:46.780 -> Voltage: 0.992 V | Current: 4.00 mA | Depth: 0.0 cm
:27:47.780 -> Voltage: 1.022 V | Current: 4.09 mA | Depth: 2.7 cm
:27:48.750 -> Voltage: 1.031 V | Current: 4.13 mA | Depth: 3.9 cm
:27:49.766 -> Voltage: 1.036 V | Current: 4.14 mA | Depth: 4.5 cm
:27:50.762 -> Voltage: 1.041 V | Current: 4.16 mA | Depth: 5.1 cm
:27:51.773 -> Voltage: 1.051 V | Current: 4.20 mA | Depth: 6.4 cm
:27:52.776 -> Voltage: 1.051 V | Current: 4.20 mA | Depth: 6.4 cm
:27:53.790 -> Voltage: 1.056 V | Current: 4.22 mA | Depth: 7.0 cm
```

Figure 3.2.2: Pressure Level Sensor Results 02

3.2.1 Observation

Table 3.2.1: Pressure Sensor Readings and Vertical Motor Response

Voltage	Current (mA)	Calculated Depth (cm)	Motor Action
0.98 V	4.0 mA	0 cm	Downward Move
1.25 V	5.0 mA	31.3 cm	Downward Move
2.5 V	10.0 mA	187.5 cm	Downward Move
3.75 V	15.0 mA	375.0 cm	Upward Move
5.0 V	20.0 mA	500.0 cm	Upward Move

Figure 3.2.3: Pressure Level Sensor In Water

3.3 Motor Control Performance

The PWM-based control provided smooth operation between 130–255 PWM range.

The system applied direction control using H-Bridge logic, enabling both clockwise and counterclockwise motion.

Both motors were able to stop precisely at the desired setpoints.

The tilt motor reset condition was confirmed under angle overflow.

No observable motor jitter was present due to the inclusion of deadband tolerances.

3.4 Serial Output Sample

Tilt Angle: 44.00° | Tilt Forward | Depth: 260.0 cm | Depth Motor Up

Tilt Angle: 0.10° | Tilt Stable | Depth: 249.0 cm | Depth Stable

Tilt Angle: 181.0° | Tilt >180°: Resetting to 0°

3.5 Observations and Conclusions

The sensor readings were stable and consistent across repeated runs. The system effectively stabilized both tilt and depth using minimal hardware. The automatic reset for tilt angle beyond 180° worked as intended The depth motor respected boundary limits (0–500 cm), preventing overrun

Chapter 4

Conclusions

This project successfully demonstrated the design and implementation of a real-time, closed-loop control system for managing the orientation and elevation of an underwater echosounder transducer. By integrating an analog tilt sensor and a current-loop-based pressure sensor with an Arduino Leonardo microcontroller, the system achieved dynamic adjustment of beam direction and submersion depth, which is essential in maintaining acoustic performance in variable water environments.

The tilt control mechanism operated within a defined 0–180° range and incorporated a self-correction routine for angles exceeding safety thresholds (±0.1 degree). Likewise, the vertical positioning system maintained target depth within a ±1 cm tolerance, aided by the stable analog-to-digital conversion of pressure readings. The combination of deadband logic, PWM-based motor control, and embedded decision-making algorithms ensured efficient and smooth motor operation while minimizing unnecessary actuation.

Overall, the system validated the core hypothesis that an affordable embedded platform can be used to automate beam alignment in underwater acoustic systems, addressing challenges of beam distortion and surface interference in shallow water environments.

4.1 Future work

While the system performed reliably in controlled settings, several improvements can enhance its robustness, precision, and field-readiness:

1. Digital Sensor Integration Replacing analog sensors with digital equivalents would reduce susceptibility to noise, improve resolution, and enable more precise control. Digital communication protocols (e.g., I2C, SPI) can improve data accuracy and reliability.

2. Closed-Loop Feedback with Encoders Incorporating rotary encoders or magnetic feedback sensors on motors would enable true positional awareness, improving control accuracy and allowing for more complex control algorithms such as PID or trajectory planning.

3. Advanced Filtering Techniques Implementing digital filtering (e.g., Kalman filter, exponential smoothing) would stabilize noisy sensor data, especially under turbulent water or variable current conditions.

4. Wireless Monitoring and Control Adding a wireless communication module (e.g., Bluetooth, LoRa, or Wi-Fi) would enable remote monitoring and control—useful in both laboratory and deployed environments.

5. Power Management and Waterproofing For real-world deployment, optimizing power consumption and implementing robust waterproofing solutions will be essential to ensure long-term autonomous operation.

6. Field Testing and Environmental Adaptation Deploying the system in real river, lake, or coastal environments will provide valuable insights into its performance under natural disturbances, such as currents, debris, temperature fluctuations, and salinity.

7. Integration with Sonar/Echosounder Systems A natural extension of this work is the integration of real-time sonar feedback to guide beam alignment, creating a fully adaptive hydroacoustic sensing system.

4.2 Declarations

1. Circuit Design: Proteus 8.0 Professional was utilized for designing the Single Line Diagram (SLD) of the system, ensuring accurate representation of electronic components and their interconnections.

2. Embedded Programming: The Arduino IDE was employed for developing and uploading C-based control logic to the microcontroller. Serial data communication and debugging were performed at a baud rate of 9600 for real-time monitoring.

3. Hardware Visualization: The hardware layout and wiring connections were illustrated using the Canvas platform to provide a clear pictorial representation of the physical setup.

4. Documentation Support: Online resources including Google and ChatGPT were used for rephrasing technical content and eliminating grammatical inaccuracies to maintain a professional standard in report writing.

5. System Design Presentation: Microsoft PowerPoint was used to create the high-level system design diagrams, providing an overview of major system components and their interactions.

Bibliography

[1] C. F. Berg, "Permeability description by characteristic length, tortuosity, constriction and porosity," *Transport in Porous Media*, vol. 103, no. 3, pp. 381–400, 2014.

[2] A. K. Doughty, B. J. Horton, T. Nguyen, C. Huyen, R. Ballagh, R. Corkrey, and G. N. Hinch, "The influence of lameness and individuality on movement patterns in sheep," *Elsevier*, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0376635717302449

[3] R. Components, "Tms22e-pkh090 tilt sensor datasheet," https://docs.rs-online.com/419a/A700000008853445.pdf, accessed July 2025.

[4] DFRobot, "Throw-in type liquid level transmitter sen0262," https://wiki.dfrobot.com/Throw-in_Type_Liquid_Level_Transmitter_SKU_KIT0139, accessed July 2025.

[5] Arduino, "Arduino leonardo board overview," https://www.arduino.cc/en/Guide/ArduinoLeonardo, accessed July 2025.

[6] "L298n dual h-bridge motor driver datasheet," https://components101.com/modules/l298n-motor-driver-module, accessed July 2025.

[7] T. Instruments, "Understanding 4-20ma current loop for sensor applications," https://www.ti.com/lit/an/slyy137/slyy137.pdf, accessed July 2025.