Robotic Sensor Systems (RSS) Project

Group Number 6

Prof. Dr.-Ing. Robert Schmitt

Laboratory for Machine Tools and Production Engineering WZL, RWTH Aachen University

Winter Semester 2024/25

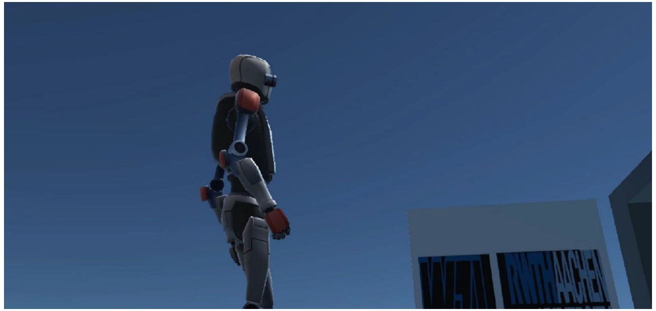
Table of Contents

Introduction

The usage of sensor systems have seen continuous growth in the industries and day-to-day lives and helped in the tasks by performing automation.

Sensors also play an important role in robotic systems by providing them with vital information such as local position, target position, perception information such as costmap and occupancy grids which help the robot in performing trajectory planning and obstacle avoidance.

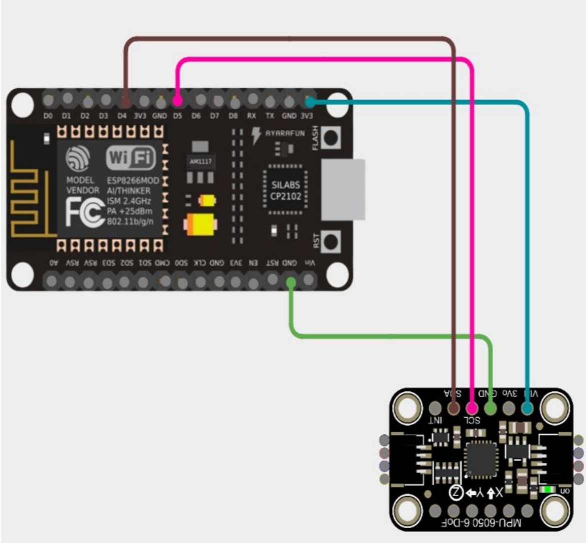
The outcome of this group project is to understand the working of various sensors and their applications by integrating the sensor system to control a humanoid robot in a predeveloped open world game environment.



Sensors

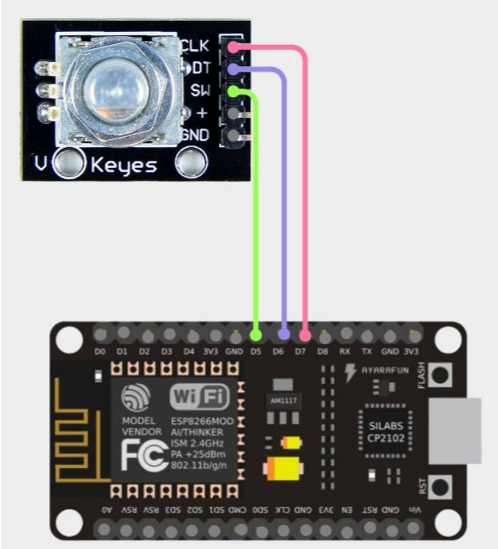
MPU6050

1. A combination of:
   a. 3-axis gyroscope
   b. 3-axis accelerometer
   c. Temperature sensor
2. Uses I2C protocol - Inter Integrated Circuit
3. Specifications:
   a. Operating Voltage: 3.3V-5V
   b. Measurement Range:
      i. Accelerometer: ±2g, ±4g, ±8g, ±16g.
      ii. Gyroscope: ±250, ±500, ±1000, ±2000 degrees/second.
   c. Arduino Library used in our project: Adafruit
4. Working Principle: Micro-electromechanical (MEMS) system for accelerometer and gyroscope and a silicon diode based temperature sensor that measures the chip temperature.



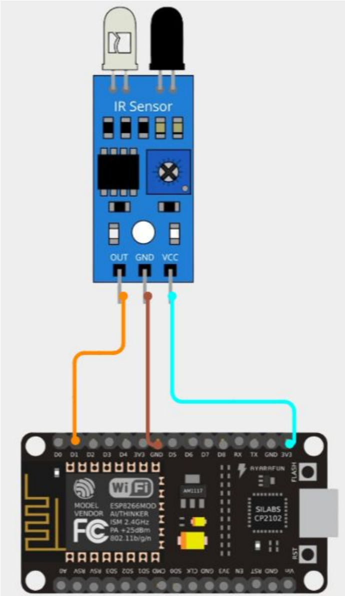
Rotary Encoder

1. An electromechanical device that converts rotational movement into an electrical signal.
2. Measures position, direction or speed.
3. Components::
   a. Rotary Shaft: Rotates to generate signals
   b. Code disk: Has patterns to encode position
   c. Sensors: Detect shaft position changes
   d. Output pins: A and B which are 90° out of phase.
4. Working Principle: Output signals from pins A and B are used to determine the direction.



Infrared Sensor

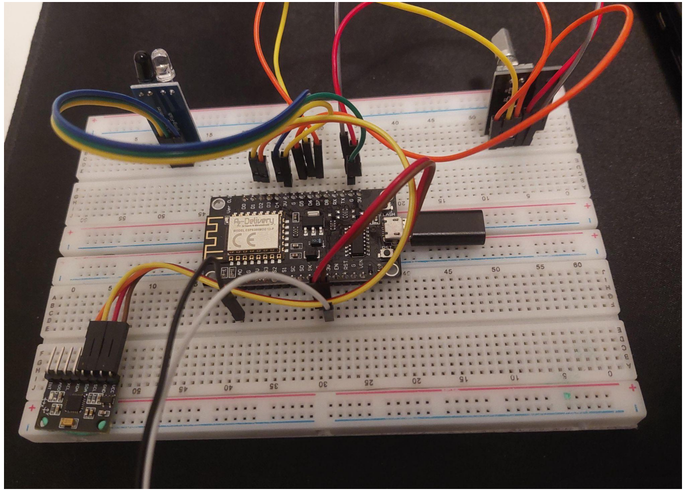
1. Detects and emits infrared radiation, operating in the infrared spectrum.
2. Used for proximity detection, object tracking, etc.
3. Components:
   a. Emitter: Infrared light
   b. Detector: Photodiode
   c. Signal Processing unit: Amplifies and processes the detected signal.
4. Working Principle: The emitter emits infrared light which gets reflected by an object. This reflected light is detected by the IR receiver. The intensity indicates proximity or presence.



Hardware Setup

Hardware Setup

- For controlling the robot, we are using the roll and pitch motion of the accelerometer.
- The roll and pitch motion govern the forward/ backward motion and the sideways motion (turning robot left and right).
- Our setup is also capable of changing the camera rotation, for which we are using the yaw motion.
- In addition to this, we are also using capacitive touch sensor to trigger the jump actions of the robot.
- The infrared sensors are used to open the doors of the arena.



Sensors and its usage in unity

Robot motion control →	Backward & forward motion	Sideways motion	Jump motion
	Using pitch angle	*Using roll angle*	*Using touch sensor*
Camera control →	Rotation (yaw)	Rotation (pitch)	Camera reset
	Using yaw angle	*Using encoder direction*	*Using encoder button*
Additional controls and sensing →	Door control	Temperature sensing	
	Using IR sensor	*Using temperature sensor*	

Logic and Explanation

Explanation of the Code

```
// Function to calibrate Gyro
void calibrateGyro() {
  Serial.println("Calibrating Gyroscope...");
  for (int i = 0; i < calibrationSamples; i++) {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
    gyroBiasX += g.gyro.x;
    gyroBiasY += g.gyro.y;
    gyroBiasZ += g.gyro.z;
    delay(5);
  }
  gyroBiasX /= calibrationSamples;
  gyroBiasY /= calibrationSamples;
  gyroBiasZ /= calibrationSamples;
  Serial.println("Gyro Calibration Complete.");
}
```

```
void calibrateAccelerometer() {
  float totalX = 0.0, totalY = 0.0, totalZ = 0.0;

  Serial.println("Calibrating Accelerometer...");

  for (int i = 0; i < calibrationSamples; i++) {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);

    totalX += a.acceleration.x;
    totalY += a.acceleration.y;
    totalZ += a.acceleration.z;

    delay(5); // Small delay between samples
  }

  // Compute offsets
  accelBiasX = totalX / calibrationSamples;
  accelBiasY = totalY / calibrationSamples;
  accelBiasZ = (totalZ / calibrationSamples) - 9.81; // Gravity adjustment for Z-axis

  Serial.println("Accelerometer Calibration Complete!");
}
```

Calibration of sensors in arduino

NodeMCU is set as a Wifi access point

```
// Start the Wi-Fi access point
WiFi.softAP(ssid, password);
Serial.print("\nAccess Point \"");
Serial.print(ssid);
Serial.println("\" started");

// Display the access point's IP address
Serial.print("IP address:\t");
Serial.println(WiFi.softAPIP());
```

Encoder function to calculate count & direction

```
void ICACHE_RAM_ATTR encoderIRS() {
  unsigned long currentInterruptTime = millis();

  // Debounce check: Ignore rapid changes within DEBOUNCE_TIME ms
  if (currentInterruptTime - lastInterruptTime < DEBOUNCE_TIME) {
      return;
  }
  lastInterruptTime = currentInterruptTime;  // Update last interrupt time
  static int lastState = 0;
  int currentState = (digitalRead(CLK_PIN) << 1) | digitalRead(DT_PIN);

  if (currentState != lastState) {
    if ((lastState == 0b00 && currentState == 0b01) ||
        (lastState == 0b01 && currentState == 0b11) ||
        (lastState == 0b11 && currentState == 0b10) ||
        (lastState == 0b10 && currentState == 0b00)) {

      counter--;
      direction = DIRECTION_CW;   // value = 1(count decrease)
    } else {
      counter++;
      direction = DIRECTION_CCW;  // value = -1(count increase)
    }
    lastState = currentState;
  }
  else{
    //Serial.println("NO_ROTATION");
    direction = 0;
  }
}
```

Explanation continued…

```
// Apply Low-Pass Filter (LPF) to accelerometer
float accX = alpha * a.acceleration.x + (1 - alpha) * prev_accX;
float accY = alpha * a.acceleration.y + (1 - alpha) * prev_accY;
float accZ = alpha * a.acceleration.z + (1 - alpha) * prev_accZ;
prev_accX = accX;
prev_accY = accY;
prev_accZ = accZ;

// Calculate angles from accelerometer (roll and pitch)
float accAngleX = atan2(accY, sqrt(accX * accX + accZ * accZ)) * 180 / PI;
float accAngleY = atan2(-accX, sqrt(accY * accY + accZ * accZ)) * 180 / PI;

// Apply Kalman filter
roll = kalmanRoll.getAngle(accAngleX, g.gyro.x * 180 / PI * elapsedTime, elapsedTime);
pitch = kalmanPitch.getAngle(accAngleY, g.gyro.y * 180 / PI * elapsedTime, elapsedTime);

// Complementary filter for yaw (no accelerometer correction)
yaw = 0.98 * (yaw + (g.gyro.z - gyroBiasZ) * elapsedTime * 180 / PI) + 0.02 * yaw;
```

```
// Store all values to be sent as a JSON
StaticJsonDocument<256> jsonDoc;
jsonDoc["ROLL"] = String(roll, 2);
jsonDoc["PITCH"] = String(pitch, 2);
jsonDoc["YAW"] = String(yaw, 2);
jsonDoc["IR"] = IRState;
jsonDoc["TEMP"] = String(temp.temperature, 2);
jsonDoc["ENCODER_COUNT"] = String(counter);
jsonDoc["ENCODER_DIR"] = String(direction);
jsonDoc["JUMP"] = jumpState;
jsonDoc["BUTTON"] = buttonState;

String jsonRes;

serializeJson(jsonDoc, jsonRes);

// Send values to Unity
client.println(jsonRes);

// Reset encoder direction only when a new movement is detected
static int lastCounter = 0;
if (lastCounter != counter) {
    lastCounter = counter; // Update last known count
    direction = 0; // Reset direction after reading the change
}
}
```

Using filters to clean the values received from Accelerometer and Gyroscope

Storing all the input values from the sensors as a json and sending this to Unity environment

Code explanation of Unity environment

```csharp
private void ConnectToESP()
{
    try
    {
        client = new TcpClient("192.168.4.1", 80); // Replace with NodeMCU IP and port
        stream = client.GetStream();
        isRunning = true;

        receiveThread = new Thread(ReceiveMessages);
        receiveThread.IsBackground = true;
        receiveThread.Start();

        Debug.Log("Connected to NodeMCU.");
    }
```

Establishes TCP client connection with NodeMCU

```csharp
private void ProcessMessage(string message)
{
    message = message.Trim();

    if (string.IsNullOrWhiteSpace(message)) return;

    try
    {
        // Parse the values and store it in a dictionary for easy access
        var parsedData = JsonConvert.DeserializeObject<Dictionary<string, string>>(message);

        if (parsedData != null)
        {
            foreach (var item in parsedData)
            {
                SensorData[item.Key] = item.Value;
            }

            Debug.Log("Processed JSON: " + message);
        }
    }
```

Parse the JSON values and store it in a dictionary for easy access across unity

Code explanation of Unity environment

```
private void CameraRotation()
{

    // float deltaTimeMultiplier = Time.deltaTime;
    float deltaTimeMultiplier = 0.25f;
    float c_yaw = 0.0f; // Declare local variables
    float c_pitch = 0.0f;

     // Check for camera reset input
    if (TCPManager.Instance != null && TCPManager.Instance.SensorData.TryGetValue("BUTTON", out string resetCam))
    {
        if (resetCam == "TRUE")
        {
            // Reset yaw and pitch to default values
            // _cinemachineTargetYaw = defaultYaw;
            _cinemachineTargetPitch = defaultPitch;

        }
    }
```

Code block to reset camera pitch angle using encode button input

```
// Update camera rotation based on TCP data
if (TCPManager.Instance != null && TCPManager.Instance.SensorData.TryGetValue("YAW", out string yaw) &&
    TCPManager.Instance.SensorData.TryGetValue("ENCODER_DIR", out string cam_pitch))
{

    // Set the yaw and pitch values based on the sensor data
    float yaw_deg = float.Parse(yaw);
    float cam_pdeg = float.Parse(cam_pitch);

    // Set the c_yaw value if the sensor data is within a certain range of degrees
    if (Math.Abs(yaw_deg) > 20 && Math.Abs(yaw_deg) < 90)
    {
        c_yaw = ((yaw_deg) > 0)? -1.0f : 1.0f;
    }

    // Set the  c_pitch value based on the enoder direction value
    else if (cam_pdeg == 1.0f)
    {
        c_pitch = -1.0f;
    }

    else if (cam_pdeg == -1.0f)
    {
        c_pitch = 1.0f;
    }

    _cinemachineTargetYaw += c_yaw * deltaTimeMultiplier;
    _cinemachineTargetPitch += c_pitch * deltaTimeMultiplier;
}
```

Code block to control camera angle using MPU yaw angle and encoder direction value

Code explanation of Unity environment

```
if (TCPManager.Instance != null &&
    TCPManager.Instance.SensorData.TryGetValue("PITCH", out string pitch) &&
    TCPManager.Instance.SensorData.TryGetValue("ROLL", out string roll))
{
    // Set the roll and pitch values based on the sensor data
    float pitch_deg = float.Parse(pitch);
    float roll_deg = float.Parse(roll);

    // Set the speed component if the sensor data is within a certain range of degrees
    if (Math.Abs(pitch_deg) >= 40)
    {

        // Set forward speed to 1.0f or -1.0f based on sign of pitch value
        accelYValue = ((pitch_deg) < 0)? -1.0f : 1.0f;
        // Set target speed based as sprint speed
        targetSpeed = SprintSpeed;
    }

    else if (Math.Abs(pitch_deg) > 20)
    {
        // Set forward speed to 0.5f or -0.5f based on sign of pitch value
        accelYValue = ((pitch_deg) < 0)? -0.5f : 0.5f;
        // Set target speed based as move speed
        targetSpeed = MoveSpeed;
    }

    if (Math.Abs(roll_deg) >= 40)
    {
        // Set sideways speed to 1.0f or -1.0f based on sign of roll value
        accelXValue = ((roll_deg) < 0)? -1.0f : 1.0f;
        // Set target speed based as sprint speed
        targetSpeed = SprintSpeed;
    }

    else if (Math.Abs(roll_deg) > 20)
    {
        // Set sideways speed to 0.5f or -0.5f based on sign of roll value
        accelXValue = ((roll_deg) < 0)? -0.5f : 0.5f;
        // Set target speed based as move speed
        targetSpeed = MoveSpeed;
    }

}
```

Code block to control robot forward and sideways speed based on the MPU roll and pitch angle

```
// Jump
if (TCPManager.Instance != null &&
TCPManager.Instance.SensorData.TryGetValue("JUMP", out string j_button))

{
    if (j_button == "TRUE" && _jumpTimeoutDelta <= 0.0f)
    {
        // the square root of H * -2 * G = how much velocity needed to reach desired height
        _verticalVelocity = Mathf.Sqrt(JumpHeight * -2f * Gravity);

        // Reset jump timeout to prevent repeated jumps
        _jumpTimeoutDelta = JumpTimeout;

        // update animator if using character
        if (_hasAnimator)
        {
            _animator.SetBool(_animIDJump, true);
        }
    }
}
```

Robot jump control based on touch sensor input

Demonstration video

Creativity aspect of our project :)

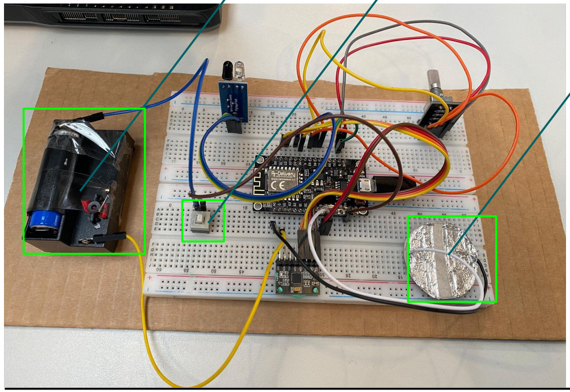
- Use of capacitive sensor to perform jumps.
- By responding to touch response, the circuit gets completed which sends Signal to jump.

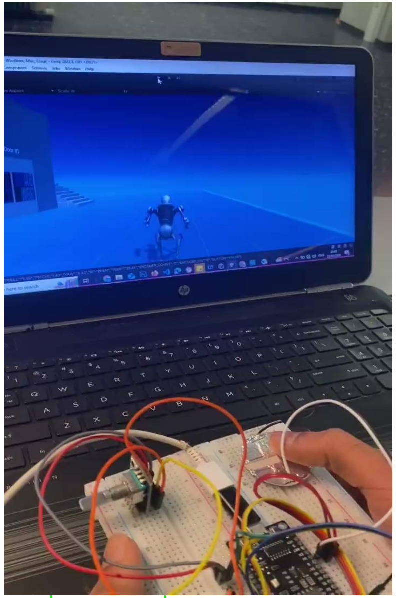
- The threshold value is in the range of ~100

Battery holder

Switch

Capacitive touch sensor





Thank you

Made by:

Abhinav
Ashwin
Karthik
Kartik
Seban