**The German University in Cairo (GUC)**
**Faculty of Media Engineering and Technology**
**Computer Science and Engineering**
**Embedded System Architecture - CSEN 701**

# Magneto Glove

*Team Members :*

**Omar Hatem, #55-7682**

**Malak Osama, #55-9108**

**Farah Khaled, #55-5083**

**Seif Wael, #55-13503**

**Kareem Sherif, #55-8812**

**Shahd Matar, #55-5683**

**Omar Seif, #55-3099**

*Team Number :*
**Team # 18**

*Under Supervision of :*
**Dr. Eng. Catherine M. Elias**

**Winter 2024**

# Contents

# Subsystems

## Subsystem 1: Tilt Detection

### Functionality:

This subsystem is responsible for:

- Acquires accelerometer and gyroscope data from the MPU6050.
- Processes the raw data to determine the tilt direction.
- Sends the tilt code to the ESP32 over I2C.

### Drivers Used:

1. **MPU6050 Driver**:
   - `mpu6050.h` and `mpu6050.c`: Handle initialization, data acquisition, and data processing for the MPU6050 sensor.
2. **I2C Communication Driver**:
   - Pico SDK's `hardware/i2c.h`: Facilitates communication between the Raspberry Pi Pico W and the ESP32 module.
   -

### Hardware Components:

| Component | Description | Datasheet Link | Driver Name |
|---|---|---|---|
| Raspberry Pi Pico W | Microcontroller | Datasheet | Pico SDK |
| MPU6050 | Accelerometer and Gyroscope Sensor | Datasheet | mpu6050.h |

### Connections:

- SDA (ESP32) → GPIO 6 (Pico W)
- SCL (ESP32) → GPIO 7 (Pico W)
- SDA (MPU) → GPIO 4
- SCL (MPU) → GPIO 5

# Subsystems

## Subsystem 2: ESP Broadcast Node

**Functionality:**

This subsystem is responsible for:

- Receives tilt codes via I2C as a slave device.
- Broadcasts received tilt codes wirelessly using ESP-NOW

**Drivers Used**

**I2C Communication Driver:**

- **Wire.h**: Handles I2C communication as a slave device to receive tilt data from the Raspberry Pi Pico W.

**ESP-NOW Communication Driver**

- **ESP32_NOW.h**: Manages wireless broadcast communication using the ESP-NOW protocol.

**Hardware Components:**

| Component | Description | Datasheet Link |
|---|---|---|
| ESP32 | Microcontroller | Datasheet |

**Connections:**

- SDA (ESP32) → GPIO 6 (Pico W)
- SCL (ESP32) → GPIO 7 (Pico W)

# Subsystems

## Subsystem 3: ESP Receiving Node

### Functionality

This subsystem is responsible for:

- Receiving data wirelessly using the ESP-NOW protocol.
- Translating the received data into binary outputs on three GPIO pins to communicate with the Arduino Nano RP2040.

### Drivers Used:

- **esp_now.h**: Handles the reception of data packets from the ESP-NOW broadcast network.
- **Arduino.h (Built-in)**: Manages GPIO pin states to represent received data as binary outputs.

| Component | Description | Datasheet Link |
|---|---|---|
| ESP32 | Microcontroller | Datasheet |

### Connections:

#### GPIO Outputs to Arduino Nano RP2040

- GPIO 25 (ESP32) → GPIO 27 (Arduino)
- GPIO 26 (ESP32) → GPIO 28 (Arduino)
- GPIO 27 (ESP32) → GPIO 29 (Arduino)

# Subsystems

## Subsystem 3: Car Control

**Functionality:**

This subsystem is responsible for:

- Receiving binary data via GPIO pins from the ESP32.
- Interpreting the binary input to determine tilt direction.
- Controlling motor movement and speed based on the tilt direction.
- Activating a buzzer and LEDs when backward tilt is detected.
- Stopping the car when an obstacle is detected by the IR sensor.

**Drivers Used:**

**Motor Control Driver:**

- **motor_driver.h**: Manages motor direction and speed.

**IR Sensor Driver:**

- **ir_driver.h**: Reads data from the IR sensor to detect obstacles.

**Hardware Components:**

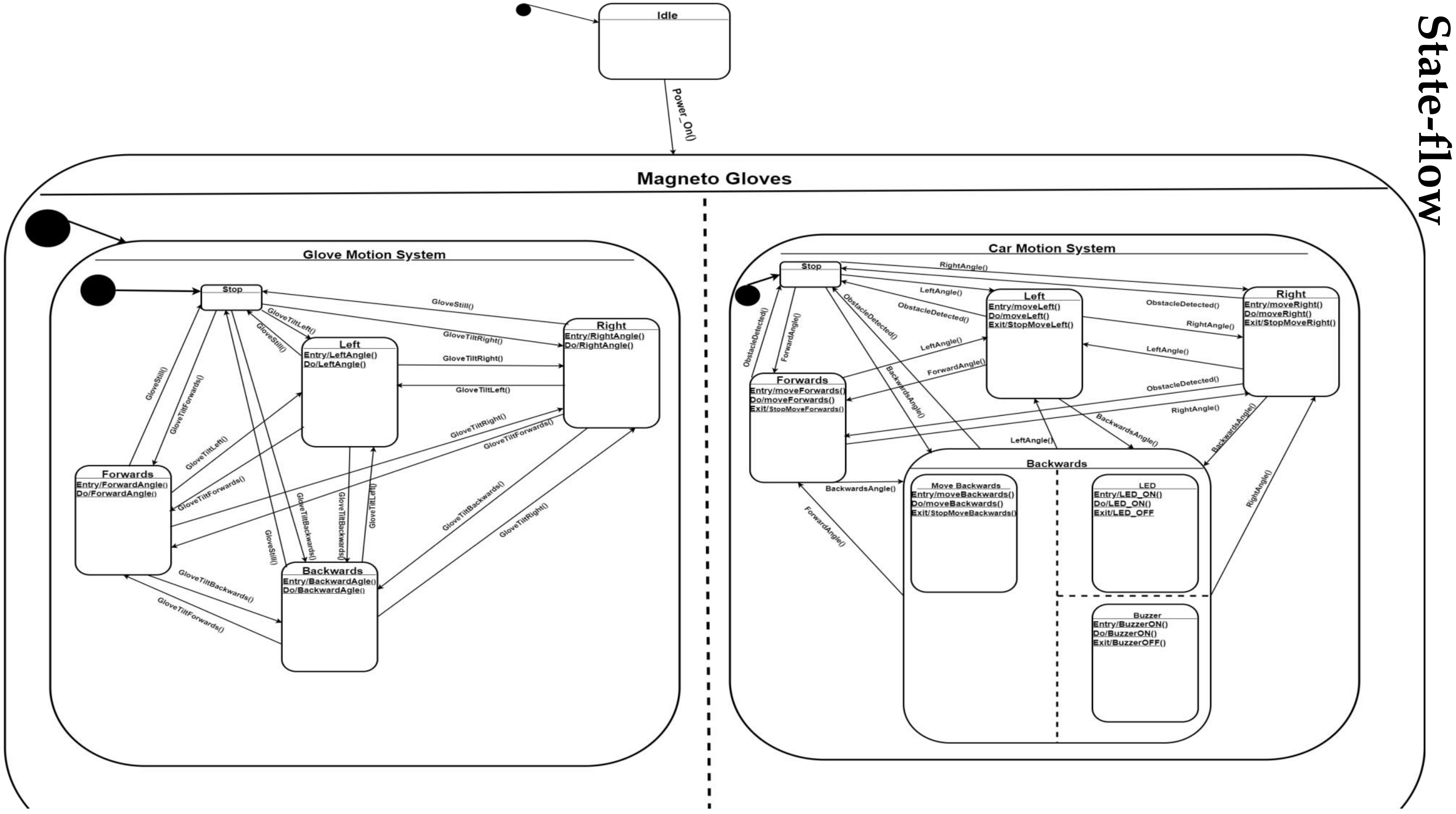
| Component | Description | Datasheet Link | Driver Name |
|---|---|---|---|
| Arduino Nano RP2040 Connect | Microcontroller | Datasheet | |
| L298N Motor Driver | Controls motor speed and direction | Datasheet | Motor_driver.h |
| IR Sensor | Obstacle detection sensor | Datasheet | ir_driver.h |
| Buzzer | Emits sound for alerts | Datasheet | |
| LEDs | Visual indicators for alerts | Datasheet | |

# Subsystems

**Connections:**

- **Input from ESP32**
    - GPIO 27 → Input Signal 1
    - GPIO 28 → Input Signal 2
    - GPIO 29 → Input Signal 3
- **Motor Driver**
    - MOTOR_A_PWM → GPIO 16
    - MOTOR_A_IN1 → GPIO 25
    - MOTOR_A_IN2 → GPIO 15
    - MOTOR_B_PWM → GPIO 19
    - MOTOR_B_IN1 → GPIO 17
    - MOTOR_B_IN2 → GPIO 18
- **IR Sensor**
    - - IR_SENSOR → GPIO 26
- **LEDs**
    - GPIO 12 → LED 1
    - GPIO 13 → LED 2
- **Buzzer**
    - GPIO 21

# State-flow



'24W - CSEN 701 - Project Team #18

# State-flow

Concurrency in this system is achieved through the division of tasks across different microcontrollers and the use of interrupt-driven and communication protocols. The logic behind the concurrency is to ensure efficient and parallel handling of various tasks, avoiding bottlenecks and ensuring real-time responsiveness. Here's how concurrency is implemented and the rationale for each part:

---

1. **Raspberry Pi Pico W**

- **Concurrency Mechanism**: I2C communication and periodic sensor polling.
- **Explanation**:
  - The Pico W reads data from the MPU6050 over I2C in a periodic loop.
  - The loop ensures non-blocking behavior by polling at a defined interval, allowing the processor to manage other tasks if needed.
  - This design ensures that fresh sensor data is always available for transmission to the ESP32 without delays.

---

2. **ESP32 (Transmitter Node)**

- **Concurrency Mechanism**:
  - **I2C communication**: Data is received from the Pico W.
  - **ESP-NOW transmission**: Handled via interrupt-driven callbacks.
- **Explanation**:
  - The ESP32 communicates with the Pico W over I2C and receives tilt codes.
  - As soon as the data is received, it triggers an interrupt or callback mechanism to transmit the data over ESP-NOW.
  - This ensures no delays in processing incoming or outgoing data, enabling a seamless flow of information.

---

# State-flow

3. **ESP32 (Receiver Node)**

- **Concurrency Mechanism**:
    - **ESP-NOW reception**: Interrupt-driven callback for data reception.
    - **Binary GPIO Output**: Controlled based on received data.
- **Explanation**:
    - The ESP32 receiver uses ESP-NOW callbacks to handle incoming data as soon as it is broadcasted by the transmitter.
    - Upon reception, it immediately processes the data and updates the GPIO pins.
    - This non-blocking behavior ensures that the ESP32 can simultaneously listen for new broadcasts while maintaining its GPIO outputs in real-time.

---

4. **Arduino Nano RP2040 Connect**

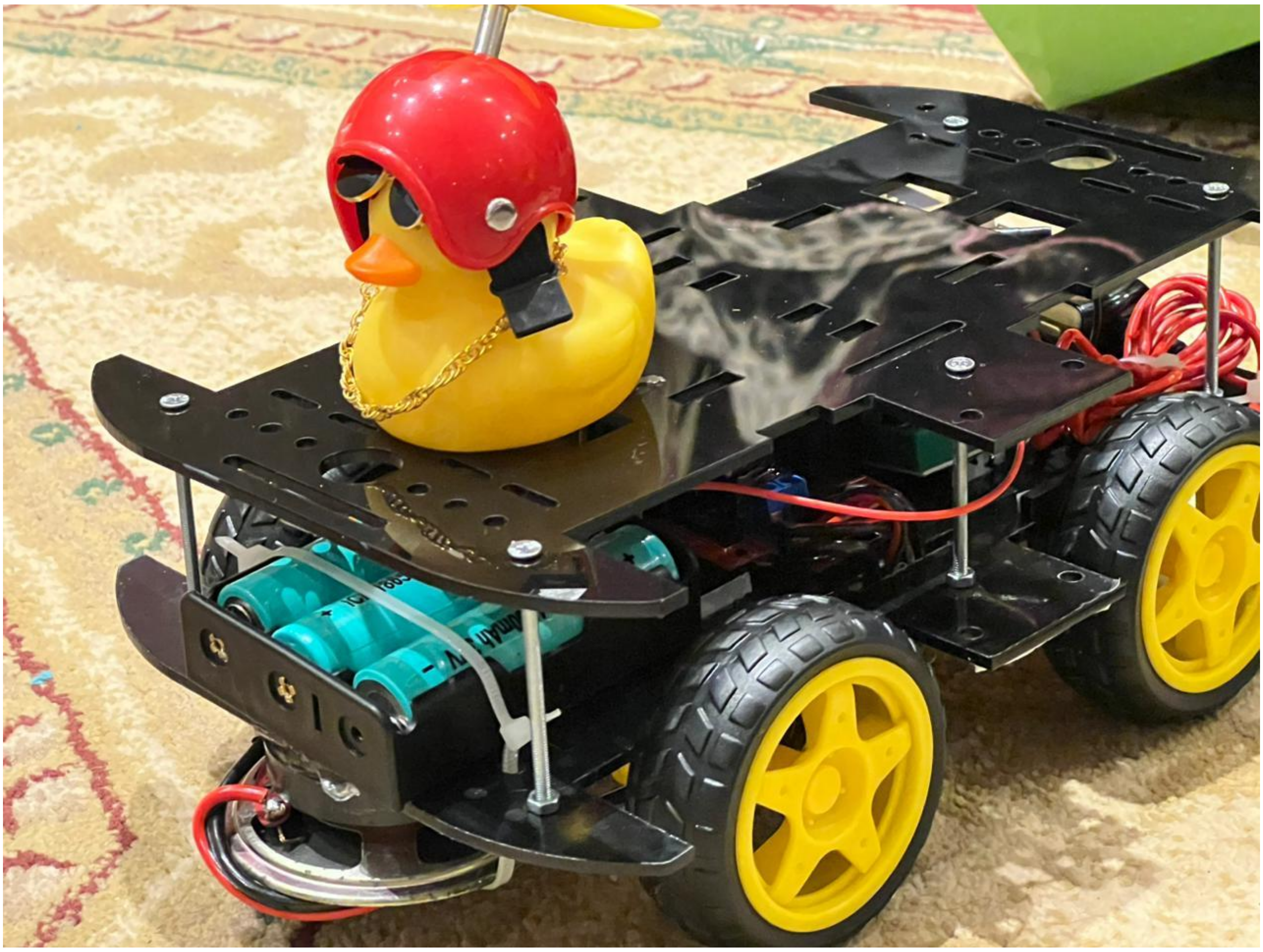
- **Concurrency Mechanism**:
    - **GPIO polling**: Reads input signals from the ESP32.
    - **Motor Control**: Executes control logic based on the inputs.
    - **Interrupt Handling**: IR sensor input interrupts for obstacle detection.
- **Explanation**:
    - The RP2040 polls the GPIO pins for binary signals from the ESP32 in a loop, interpreting the signals into tilt directions.
    - Concurrently, motor control logic adjusts speed and direction based on the interpreted tilt data.
    - The IR sensor operates on an interrupt-driven mechanism, ensuring that obstacle detection can immediately override other tasks (e.g., halting the motor) without delay.

Wdw

# Github Repo Link

Wdw

# Github Repo Link



Wdw

# Github Repo Link

**https://github.com/Omar-Hatem33/MS_01_Team_18_Project.git**