Trinity Dhillon

CSC 155

Professor Re

12/3/2024

Problems	Solutions
Initializing I2C Communication. issues with communication between the Raspberry Pi and the I2C devices (PWM drivers and Motor HAT).	ensure that the I2C interface was properly enabled in the Raspberry Pi's settings. I had to edit the Raspberry Pi's config file to enable the I2C interface: sudo raspi-config
Motors Not Running motors were not responding to commands. Despite correctly wiring the motors and initializing the Motor HAT, the motor control commands (run(), setPin(), etc.) seemed to have no effect on the motors.	incorrect initialization of the Motor HAT library and misconfigured GPIO pins for motor control. The Motor HAT required configuration of the I2C bus and the initialization of motors. I ensured that the motor pins (IN1, IN2, PWM) were properly configured setPin(motors[0].pwm, 255); // PWM to full speed

	setPin(motors[0].in1, 1); // IN1 high
	setPin(motors[0].in2, 0); // IN2 low
	run(FORWARD, 0); // motor 0 to move forward

(Undefined References) multiple .c files, such as a robot project with files like MotorHat.c, PWM.c, I2C.c, Sensor.c, and Robot.c, the linker failed to find certain functions	Had to Verify that all functions and variables declared in header files were properly defined in the corresponding .c file Biggest fix: runSensor() was declared in Sensor.h, ensured that there was corresponding definition in Sensor.c

Incorrect Function Prototypes or Mismatched Parameters
 	
MotorHat.h declared setSpeed(int speed) but MotorHat.c defines it as setSpeed(float speed), error
	match the function definitions in the .c file exactly, including parameter types
	
	// MotorHat.h
	void setSpeed(int speed);
	
	// MotorHat.c
	void setSpeed(int speed) {
	  // Function body
	}

the sensor was intended to stop the motors when an obstacle was detected, but the sensor readings were sometimes erratic, leading to unexpected behavior.	check the sensor reading before deciding whether to move the motors forward or stop them. If the distance returned by the sensor was below a certain threshold the motors would stop.
	
	if (distance < 10) {
	  stopMotors();
	} else {
	  forward();
	}
	
	ensured that the sensor was read at regular intervals during motor movement, allowing the robot to respond to obstacles in its path.

PWM Driver Initialization Fails Randomly on Startup	Added checks for I2C bus availability and ensured proper deinitialization of PWM driver during shutdown.
GPIO Pin Initialization Failure	Configured all GPIO pins correctly. Used bcm2835_gpio_fsel() and bcm2835_gpio_write() functions properly.

	Added GPIO initialization code.
Issue Running for Extended Periods	Was a memory leaks. corrected dynamic memory allocation in `Robot.c`.
Random Program Crashes During I2C Operations	Added error handling and recovery in write8() and readU8() for I2C communication failures.
Compiler Warnings Regarding Implicit Declarations	Included necessary header files and ensured all function prototypes were declared before use. Fixed include statements.
Structure Issues (Multiple Definitions of main	The linker throws an error due to multiple definitions of main, indicating the function is defined in multiple files. Created a main for the sensor.c that had its own main
Code Cleanup and Refactoring Issues	Refactored code into modules for motor control, sensor reading, and PWM handling. Improved readability with comments. Added comments and function documentation.
Unresponsive Sensor After Multiple Readings	Added delays between readings.

Segmentation Fault Issues access invalid memory or dereference a NULL pointer.	Not sure how to fix!