SpotMicro Minimalist Project - Egypt (Detailed)

Executive Summary

This proposal outlines a simplified SpotMicro quadruped robot focused on three core features: obstacle avoidance, movement primitives, and a remote controller interface. The project will be built in Cairo, Egypt, using locally available components, with a total hardware budget of **6,814 EGP**.

Core Features

1. **Autonomous steering:** Utilizes an ultrasonic sensor as well as stereo cameras to be able to steer autonomously while dodging obstacles
2. **Controller Interface:** Allows for remote operation via a physical RC controller or MQTT commands Web.

Hardware Components

Mechanical Platform

- **Model:** SpotMicro V4 (Community Model from CrealityCloud: https://www.crealitycloud.com/model-detail/spotmicro-robotic-dog-v4)
- **Materials:** PLA + 3D-printed parts.
- **Printer:** Creality Cr6
- **Infill:** 20%
- **Colors:** Gray
- **Actuation:** Twelve high-torque MG996R servo motors (three per leg) provide comprehensive control over the robot's locomotion.

Electronics & Control

Component	Model	Price (EGP)	Function & Description	Source (Link)
Microcontroller	Arduino Nano RP2040 Connect	1,750	Main control board with onboard Wi-Fi for MQTT communication and an IMU for balance.	RAM Electronics: Link

Microcontrioller 2	Rasberiery Pi Pico RP2040		Secondary control, backup to the main system	
Servo Driver	PCA9685 16-Channel PWM	210	I²C-controlled driver for precise, synchronized control of all twelve servos.	MakerSelectronics: Link
Servos	MG996R Metal-Gear (×12)	2,520	High-torque servos providing powerful and precise leg actuation for dynamic movement.	MakerSelectronics: Link
Ultrasonic Sensor	HC-SR04	60	Provides distance measurement for forward obstacle detection and avoidance.	MakerSelectronics: Link
Stereo Cameras	ESP CAM32 OV2640(×2)	700	A pair of cameras used for depth	RAM Electronics: Link

			perception	
ESP32-CAM-MB MICRO USB Download Module for ESP32 CAM	(x2)		Help upload code from the computer to the cameras	
Battery	12V Li-ion 7000mAh	1,200	Powers the entire system, including the microcontroller and all twelve servos.	(Generic Estimate)
Display	2x16 line LCD display I2C	170	Small screen for providing real-time debug information and status feedback.	
DC to DC step down BUCK converter	LM2596S	50	Power the logic microcontroller 12V down to 5V	https://www.ram-e-shop.com/shop/dc-dc-100-dc-dc-step-down-voltage-converter-adjustable-3a-lm2596s-4-38vdc-to-1

				-5-35vdc-sku-dc100-7257
Step Down Buck Converter Power Supply XL4016 PWM Adjustable 4-40V To 1.25-36V Step-Down Board Module XH-M401	XL4016 PWM	150	12V down to 6V for your 12 servos.	https://www.ram-e-shop.com/shop/dc-dc-105-dc-dc-step-down-voltage-converter-adjustable-8a-xl4016-4v-40vdc-to-1-25v-36vdc-sku-dc105-7670
Rocker Switch (10A)	Switch	4	To switch on and off the robot	https://makerselectronics.com/product/rocker-switch-spdt-2-position-on-off/
Small 6V DC Motor (130-size)	Micro Metal 6V DC geared motor 30RPM GA12-N20	50	Used to actuate a simple "wagging tail" for feedback.	

H-Bridge L298N Dual H-Bridge Motor Driver	L298N Dual H-Bridge Motor Driver			
Total		**6,814 EGP**		

3. Subsystems & Operational Logic

This project is broken into four main subsystems, designed to meet the course requirements (3 subsystems, 2 parallel).

- **Subsystem 1:** Core Balance & Locomotion (Sequential)
- **Subsystem 2:** Autonomous Navigation (Parallel)
- **Subsystem 3:** Remote MQTT Control (Parallel)

Subsystems 2, and 3 are designed to run in parallel. The robot can be autonomously navigating (Sub 2) while simultaneously listening for MQTT command (Sub 3).

Subsystem 1: Core Balance & Locomotion (Sequential)

- ***Story Overview:*** *The robot's core function is to maintain balance while standing or walking, using an IMU to correct its posture.*
- ***Inputs:*** `LSM6DSOXTR 6-axis IMU` *(Analog-style data stream).*
- ***Outputs:*** `12x MG996R Servo Motors`*,* `SSD1306 OLED Display`*.*
- ***Overall Logic:***
    1. ***Initiation:*** *Robot is powered on via the* `Rocker Switch`*. The OLED Display shows "BOOTING...".*
    2. ***Calibration:*** *The system initializes the* `LSM6DSOXTR IMU`*. The OLED displays "CALIBRATING IMU... DO NOT MOVE".*
    3. ***Ready State:*** *After 3 seconds, calibration is complete. The robot moves to a default "STAND" pose. The OLED displays "STATUS: IDLE".*
    4. ***Sequential Loop (Balancing):*** *The system continuously runs this loop:*
        - *Read the current* `pitch` *and* `roll` *from the IMU.*
        - ***IF*** `|pitch| > 10` **OR** `|roll| > 10` ***degrees:*** *The system calculates*

the micro-adjustments needed for the 12 leg servos to counteract the tilt and restore balance.
- *The `PCA9685` sends the new commands to the servos.*

5. ***Command Execution:** When a command is received from another subsystem (e.g., "WALK_FWD"), this subsystem executes the gait cycle, while the "Sequential Loop (Balancing)" continues to run to ensure stability during movement.*

Subsystem 2: Autonomous Navigation (Parallel)

- ***Story Overview:** In "Patrol Mode," the robot uses its sensors to navigate a space, avoiding simple (walls).*
- ***Inputs:** `HC-SR04 Ultrasonic Sensor`, `2x OV2640 Stereo Cameras 3Mps`, `LSM6DSOXTR 6-axis IMU`*
- ***Outputs:** `I2C LCD  2x16 Display`, (Sends commands to Subsystem 1).*
- ***CV:** Stereo vision depth mapping for complex obstacle/drop-off detection.*
- ***Overall Logic:***
  1. ***Activation:** The robot receives a "PATROL" command (from Subsystem 3 or 4). The OLED displays "STATUS: PATROLLING".*
  2. ***Default State:** The system sends a "WALK_FWD" command to Subsystem 1.*
  3. ***Sensing Loop (Parallel):** The system continuously runs two sensing tasks at the same time:*
     - ***Task A (Ultrasonic):** The `HC-SR04` continuously pings for immediate frontal obstacles.*
     - ***Task B (Stereo Vision):** The `2x OV2640 3Mps Cameras` continuously process frames to build a basic depth map of the environment,*
     - ***Task C (IMU):** state estimation to check where the robot is*
  4. ***Conditional Logic (Obstacle Detected):***
     - ***IF `Stereo Vision detects a drop-off`:** This is a high-priority interrupt. The system sends "EMERGENCY_STOP" then "TURN_180" to Subsystem 1. The OLED displays "DANGER: DROP-OFF!".*
     - ***IF `HC-SR04 distance < 20cm` (Simple Wall):** The system sends "STOP" then "TURN_LEFT_90" to Subsystem 1. The OLED displays "STATUS: AVOIDING".*
  5. ***Resumption:*** `After a "TURN_LEFT_90" command, the IMU (Task C) verifies the robot has turned ~90 degrees before returning to "WALK_FWD".`

Subsystem 3: Remote MQTT Control (Parallel)

- ***Story Overview:** The robot can be fully controlled and monitored remotely over Wi-Fi via an MQTT dashboard, allowing for high-level commands and status checks. WEB0*
- ***Inputs:** `Arduino Nano RP2040 Connect` (Wi-Fi Module).*

- ***Outputs:*** *(Sends commands to all other Subsystems).*
- ***Web Interface:*** *Fulfills the "Web Interface" requirement via MQTT protocol.*
- ***Overall Logic:***
    1. ***Initiation:*** *On boot (after Subsystem 1 is "IDLE"), the* `RP2040` *connects to the designated Wi-Fi network and MQTT broker. The OLED displays "MQTT CONNECTED".*
    2. ***Listening Loop (Parallel):*** *The system subscribes to the* `spotmicro/command` *topic and listens in the background.*
    3. ***Command Execution:***
        - ***IF*** `Message == "PATROL"`***:*** *Activates Subsystem 2.*
        - ***IF*** `Message == "STOP"`***:*** *Sends "STOP" to Subsystem 1.*
        - ***IF*** `Message == "TAIL_WAG"`***:*** *Triggers the DC motor.*
        - ***IF*** `Message == "STATUS_REQUEST"`***:*** *The robot gathers data (IMU status, Obstacle status) and publishes it to the* `spotmicro/status` *topic.*
        - *IF Message == “TURN_LEFT_X”: The robot will turn left with the value X degrees*
        - *IF Message == “TURN_RIGHT_X” The robot will turn right with the value X degrees*
        - *IF Message == “WALK_FWD” The robot will move forward*
        - *IF Message == “WALK_BWD” The robot will move backward*

4. Drivers

Ultrasonic Sensor

```
/** ifndef HC_SR04_Hdefine HC_SR04_H

#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "hardware/timer.h"


 * @file hc_sr04.h
 * @brief HC-SR04 Ultrasonic Distance Sensor Driver
 *
 * Driver for HC-SR04 ultrasonic distance sensor module.
 * Supports multiple sensors on different GPIO pins.
 *
 * Typical Usage:
 * - Power: 5V (tolerates 3.3V)
 * - TRIG pin: GPIO output, pull LOW between measurements
 * - ECHO pin: GPIO input, measures pulse width (pulse_width_us / 58 =
distance_cm)
 *
 * Measurement procedure:
 * 1. Set TRIG LOW for 2us
 * 2. Set TRIG HIGH for 10us (minimum)
 * 3. Set TRIG LOW
 * 4. Measure ECHO pulse width (typically 150-25000us)
 * 5. Distance = pulse_width_us / 58 (cm) or / 148 (inches)
 */

/**
 * @brief Maximum measurement timeout in microseconds
 * HC-SR04 max range ~4m, which is ~235ms at speed of sound
 * Using 30ms (30000us) as safety timeout
 */
#define HC_SR04_TIMEOUT_US 30000
```

```c
/**
 * @brief HC-SR04 sensor configuration structure
 */
typedef struct {
    uint gpio_trig;          ///< Trigger pin (GPIO output)
    uint gpio_echo;          ///< Echo pin (GPIO input)
    char name[32];           ///< Sensor identifier (e.g., "front", "left")
    uint64_t last_pulse_us;  ///< Last measured pulse width in microseconds
    float last_distance_cm;  ///< Last measured distance in centimeters
    bool initialized;        ///< Initialization status flag
} hc_sr04_t;

/**
 * @brief Initialize HC-SR04 sensor
 *
 * Configures GPIO pins for TRIG (output) and ECHO (input).
 * Sets up the sensor structure.
 *
 * @param sensor Pointer to hc_sr04_t structure
 * @param gpio_trig Trigger pin (GPIO number)
 * @param gpio_echo Echo pin (GPIO number)
 * @param name Sensor name (copied into structure, max 31 chars)
 * @return true if initialization successful, false otherwise
 */
bool hc_sr04_init(hc_sr04_t *sensor, uint gpio_trig, uint gpio_echo, const
char *name);

/**
 * @brief Trigger a measurement cycle
 *
 * Sends 10us pulse on TRIG pin to start measurement.
 * Must wait ~60ms before reading results (see hc_sr04_read).
 *
 * @param sensor Pointer to hc_sr04_t structure
 * @return true if trigger sent successfully
 */
bool hc_sr04_trigger(hc_sr04_t *sensor);

/**
 * @brief Read measurement results from last trigger
 *
 * Waits for ECHO pin to go HIGH, then measures pulse width.
 * Timeout if ECHO doesn't respond within HC_SR04_TIMEOUT_US.
```

```
 *
 * Typical timing:
 * - ECHO goes HIGH ~200us after TRIG pulse
 * - Pulse duration: 150us (2cm) to 25000us (430cm)
 *
 * @param sensor Pointer to hc_sr04_t structure
 * @return true if measurement successful, false if timeout
 */
bool hc_sr04_read(hc_sr04_t *sensor);

/**
 * @brief Get last measured distance in centimeters
 *
 * @param sensor Pointer to hc_sr04_t structure
 * @return Distance in centimeters (0.0 if no valid measurement)
 */
float hc_sr04_get_distance_cm(hc_sr04_t *sensor);

/**
 * @brief Get last measured distance in inches
 *
 * @param sensor Pointer to hc_sr04_t structure
 * @return Distance in inches
 */
float hc_sr04_get_distance_inch(hc_sr04_t *sensor);

/**
 * @brief Get last measured pulse width
 *
 * @param sensor Pointer to hc_sr04_t structure
 * @return Pulse width in microseconds
 */
uint64_t hc_sr04_get_pulse_us(hc_sr04_t *sensor);

/**
 * @brief Perform complete measurement (trigger + read)
 *
 * Convenience function that triggers measurement and immediately reads
result.
 * WARNING: This is a blocking call that takes ~60ms total!
 * For non-blocking operation, call hc_sr04_trigger() then hc_sr04_read()
separately.
 *
```

```
 * @param sensor Pointer to hc_sr04_t structure
 * @return true if measurement successful, false if timeout
 */
bool hc_sr04_measure(hc_sr04_t *sensor);

/**
 * @brief Get sensor name
 *
 * @param sensor Pointer to hc_sr04_t structure
 * @return Pointer to sensor name string
 */
const char* hc_sr04_get_name(hc_sr04_t *sensor);

#endif // HC_SR04_H
```

LCD:

```
#ifndef LCD_16X2_H
#define LCD_16X2_H

#include "pico/stdlib.h"
#include "hardware/i2c.h"

// LCD I2C Address (common addresses)
// Check with I2C scanner if not working
#define LCD_I2C_ADDRESS 0x3F  // Most common address (PCF8574)
// Alternative addresses: 0x3F, 0x20, 0x21

// LCD Commands
#define LCD_CLEARDISPLAY 0x01
#define LCD_RETURNHOME 0x02
#define LCD_ENTRYMODESET 0x04
#define LCD_DISPLAYCONTROL 0x08
#define LCD_CURSORSHIFT 0x10
#define LCD_FUNCTIONSET 0x20
#define LCD_SETCGRAMADDR 0x40
#define LCD_SETDDRAMADDR 0x80
```

```c
// Entry Mode Set bits
#define LCD_ENTRYRIGHT 0x00
#define LCD_ENTRYLEFT 0x02
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00

// Display Control bits
#define LCD_DISPLAYON 0x04
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
#define LCD_CURSOROFF 0x00
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

// Function Set bits
#define LCD_8BITMODE 0x10
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
#define LCD_1LINE 0x00
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00

// PCF8574 Port bits (for I2C backpack)
#define LCD_BACKLIGHT 0x08
#define LCD_ENABLE 0x04
#define LCD_RW 0x02
#define LCD_RS 0x01

// LCD structure
typedef struct {
    i2c_inst_t *i2c_port;
    uint8_t address;
    uint8_t cols;
    uint8_t rows;
    uint8_t backlight_state;
} lcd_t;

// Function prototypes
bool lcd_init(lcd_t *lcd, i2c_inst_t *i2c_port, uint sda_pin, uint scl_pin,
              uint8_t cols, uint8_t rows, uint8_t address);
void lcd_clear(lcd_t *lcd);
void lcd_home(lcd_t *lcd);
void lcd_set_cursor(lcd_t *lcd, uint8_t col, uint8_t row);
```

```c
void lcd_print(lcd_t *lcd, const char *str);
void lcd_print_char(lcd_t *lcd, char c);
void lcd_backlight_on(lcd_t *lcd);
void lcd_backlight_off(lcd_t *lcd);
void lcd_display_on(lcd_t *lcd);
void lcd_display_off(lcd_t *lcd);
void lcd_cursor_on(lcd_t *lcd);
void lcd_cursor_off(lcd_t *lcd);
void lcd_blink_on(lcd_t *lcd);
void lcd_blink_off(lcd_t *lcd);
void lcd_scroll_left(lcd_t *lcd);
void lcd_scroll_right(lcd_t *lcd);
void lcd_print_int(lcd_t *lcd, int value);
void lcd_print_float(lcd_t *lcd, float value, int decimals);

// Custom character support
void lcd_create_char(lcd_t *lcd, uint8_t location, const uint8_t
charmap[8]);

#endif // LCD_16X2_H
```

Servo Driver

```c
#ifndef PCA9685_H
#define PCA9685_H

#include "pico/stdlib.h"
#include "hardware/i2c.h"

// PCA9685 Default I2C Address
#define PCA9685_ADDRESS 0x40

// PCA9685 Registers
#define PCA9685_MODE1 0x00
#define PCA9685_MODE2 0x01
#define PCA9685_SUBADR1 0x02
#define PCA9685_SUBADR2 0x03
#define PCA9685_SUBADR3 0x04
#define PCA9685_PRESCALE 0xFE
```

```c
#define PCA9685_LED0_ON_L 0x06
#define PCA9685_LED0_ON_H 0x07
#define PCA9685_LED0_OFF_L 0x08
#define PCA9685_LED0_OFF_H 0x09
#define PCA9685_ALL_LED_ON_L 0xFA
#define PCA9685_ALL_LED_ON_H 0xFB
#define PCA9685_ALL_LED_OFF_L 0xFC
#define PCA9685_ALL_LED_OFF_H 0xFD

// MODE1 bits
#define MODE1_RESTART 0x80
#define MODE1_SLEEP 0x10
#define MODE1_ALLCALL 0x01
#define MODE1_AI 0x20

// Servo configuration
#define SERVO_MIN_PULSE 500   // Minimum pulse width in microseconds (0°)
#define SERVO_MAX_PULSE 2500  // Maximum pulse width in microseconds (180°)
#define SERVO_FREQUENCY 50    // Standard servo frequency 50Hz (20ms
period)

// PCA9685 structure
typedef struct {
    i2c_inst_t *i2c_port;
    uint8_t address;
    uint sda_pin;
    uint scl_pin;
    uint16_t pwm_frequency;
} pca9685_t;

// Servo structure
typedef struct {
    pca9685_t *controller;
    uint8_t channel;        // 0-15
    uint16_t min_pulse;     // Minimum pulse width in microseconds
    uint16_t max_pulse;     // Maximum pulse width in microseconds
    uint16_t min_angle;     // Minimum angle (typically 0)
    uint16_t max_angle;     // Maximum angle (typically 180)
} servo_t;

// Function prototypes - PCA9685
bool pca9685_init(pca9685_t *pca, i2c_inst_t *i2c_port, uint sda_pin, uint
scl_pin, uint8_t address);
```

```c
void pca9685_reset(pca9685_t *pca);
void pca9685_set_pwm_freq(pca9685_t *pca, uint16_t freq);
void pca9685_set_pwm(pca9685_t *pca, uint8_t channel, uint16_t on, uint16_t
off);
void pca9685_set_all_pwm(pca9685_t *pca, uint16_t on, uint16_t off);
void pca9685_sleep(pca9685_t *pca);
void pca9685_wakeup(pca9685_t *pca);

// Function prototypes - Servo
void servo_init(servo_t *servo, pca9685_t *controller, uint8_t channel,
                uint16_t min_pulse, uint16_t max_pulse,
                uint16_t min_angle, uint16_t max_angle);
void servo_set_angle(servo_t *servo, float angle);
void servo_set_pulse(servo_t *servo, uint16_t pulse_us);
void servo_disable(servo_t *servo);

// Utility functions
uint16_t servo_angle_to_pulse(servo_t *servo, float angle);
float servo_pulse_to_angle(servo_t *servo, uint16_t pulse_us);

#endif // PCA9685_H
```

H Bridge

```c
#ifndef H_BRIDGE_L298N_H
#define H_BRIDGE_L298N_H

#include "pico/stdlib.h"

// Motor direction definitions
typedef enum {
    MOTOR_STOP = 0,
    MOTOR_FORWARD,
    MOTOR_BACKWARD,
    MOTOR_BRAKE
} motor_direction_t;

// Motor structure
typedef struct {
    uint in1_pin;
```

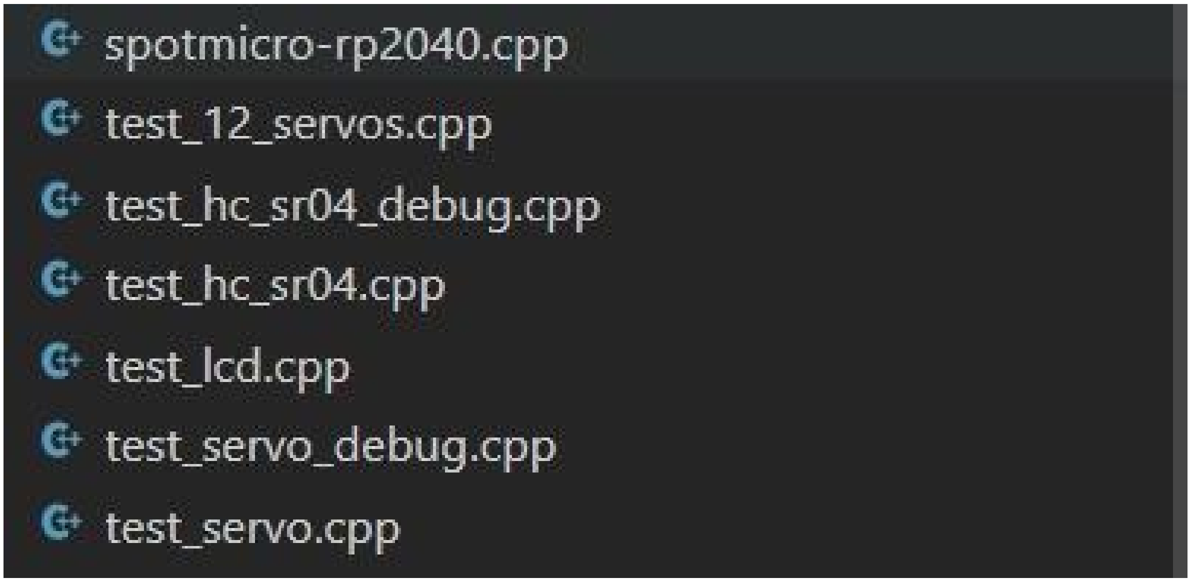
```
    uint in2_pin;
} motor_t;

// Function prototypes
void motor_init(motor_t *motor, uint in1_pin, uint in2_pin);
void motor_set_direction(motor_t *motor, motor_direction_t direction);
void motor_stop(motor_t *motor);
void motor_brake(motor_t *motor);

#endif
```

Project Directory:

```
.gitignore
blink.pio
CMakeLists.txt
h_bridge_l298n.cpp
h_bridge_l298n.h
hc_sr04.cpp
hc_sr04.h
lcd_16x2.cpp
lcd_16x2.h
pca9685.cpp
pca9685.h
pico_sdk_import.cmake
SpotMicro Minimalist Project .docx.pdf
spotmicro-rp2040.cpp
test_12_servos.cpp
test_hc_sr04_debug.cpp
test_hc_sr04.cpp
test_lcd.cpp
test_servo_debug.cpp
test_servo.cpp
```



STATE CHART ==vvv

SpotMicro Robot System - Statechart Model (v2)

Updated with Recovery Transition