*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

CHAPTER 3

RoboKar Project Technical Paper

Khoo Ann Hong, Lai Zhi Yuan, Nur Amirah Athirah Binti Mohd.Nazli, Nur Farra Aqila Binti Roslee (Group CodeSync)

RoboKar: Real-Time Embedded Robot Using MicroC/OS-II
Edited by Group CodeSync
©2018 Penerbit UTM Press, Universiti Teknologi Malaysia

3.1 Introduction

The era of automation and embedded systems has turned autonomous mobile robots into an essential element of smart and responsive environments. One such application is the RoboKar — a real-time, line-following robot would be able to follow a predetermined path (track) autonomously and in real-time without direct human control to react to environmental stimuli such as obstacles and light sources. In this technical paper, the design, implementation, and testing of the RoboKar system developed by **Group CodeSync** was carried out as a part of the course work related to the Real-Time Software Engineering (RTSE) course.

The implementation of the RoboKar project is organised in two stages. In **phase 1**, it will concentrate on the parallel development of software with the use of the μC/OS-II Real-Time Operating System (RTOS), with the most important setup capabilities such as navigation, collision detection, and motors being implemented in different actual functions in parallel through multitasking. **Phase 2** tests the dynamic environment where the robot has to use real-time responsiveness, accuracy and path efficiency amongst others to successfully cross checkpoints and other tasks such as light and obstacle response set-up.

Our RoboKar has 4 sensors to intelligently interact with the surrounding environment: there are 2 line sensors used to keep track of the path, 2 proximity sensors used to avoid obstacles, and one light sensor that triggers certain reactive behaviors. They are modularly written according to the digital principles of RTOS in order to be deterministic, to have low latency, and to have time-sensitive events under control.

To provide the practical development with reinforcement, this paper also references contemporary research in mobile robot navigation. In particular, it discusses methods like the Curve-aware Pure Pursuit (C-PP) equation and actuator-based steering systems to consider what advancements could be made and adjust our solution to industrial standards.

The remainder of this paper includes the background of the real-time and concurrent systems, the architecture of our RoboKar, testing and performance measurement, and at last, the spots to re-develop, and improve based on both observed behavior and academic research.

3.2 Background Study

The study of mobile autonomous robotics has become very popular in universities and companies since research focuses on the areas of logistics, surveillance, agriculture, and education. The development of such systems with some combination of hardware (motors, sensors, actuators) and software (control logic, task scheduling, decision-making algorithms), which is often subject to real-time requirements. The capability of a robot to act without control, but in response to responding to the environment in a timely manner is pivotal in attaining confidence in performance.

In embedded systems, **real-time responsiveness** is achieved through deterministic software behavior, which is often governed by a **Real-Time Operating System (RTOS)**. RTOS also offers such features as preemptive multitasking, prioritizing of the tasks, inter-task communication and time regulation, which are all vital in the designing of responsive robots. In this project, the RoboKar is powered by **µC/OS-II**, a lightweight multitasking real-time operating system that supports multitasking without the heavy overhead normally associated with operating systems, and is therefore extremely suitable to run real-time applications in limited RAM microcontrollers such as the Atmel ATmega328P.

Our RoboKar is designed to operate in the game condition with a track with curves, checkpoints (A-F), and two event-based tasks (L1 and L2). The robot must:

- Follow a black line using IR line sensors.
- Detect and respond to light with honking and LED blinking (L1 task).
- Identify obstacles using proximity sensors and reverse when necessary (L2 task).

This configuration requires a precise coordination of various sensing and actuation processes, which are both to go simultaneously and be responsive to real-time results. Thus, concurrent programming and design of RTOS are important concepts in the construction of a useful robot control system.

Moreover, in enhancing the system and drawing inspiration from modern research, this project refers to a study by Manikandan and Kaliyaperumal (2023) on *curve-aware navigation algorithms* and *actuator-based steering mechanisms*. Their work, which proposes a hardware-software co-design for handling curved paths in real-world terrain, offers valuable insights into improving motion stability and path-tracking accuracy. Though our RoboKar is a simpler system, concepts such as real-time path adaptation and curvature-sensitive speed control can inform future iterations of our design

3.2.1 Concurrent Programming

Concurrent programming is a programming paradigm where two or more tasks or processes run in parallel, or at least, seem to concurrently, by sharing processor time on one processor or by assigning different actions to different cores. In the context of embedded systems and robotics, concurrent programming is necessary for managing various time-sensitive tasks that must run in parallel, such as sensor monitoring, motor control, obstacle detection, and user interaction.

The conventional sequential programming has a function that runs sequentially. The method does not fit applications in robotic systems where different components must respond to real-world events in real time. For example, if a robot is travelling along a line and hits an obstacle, it must have the ability to interrupt its movement to take corrective action. If tasks are not handled concurrently, there could be significant delays or failures in behavior, potentially leading to missed checkpoints or collisions.

To implement concurrency in embedded systems, developers often rely on a **Real-Time Operating System (RTOS)**. An RTOS enables the execution of multiple tasks using mechanisms such as:

- **Preemptive multitasking**: Allows higher-priority tasks to interrupt lower-priority ones.

- **Task prioritization**: Ensures time-critical functions (e.g., collision detection) are not delayed by less critical ones (e.g., LED blinking).

- **Inter-task communication**: Manages task coordination using shared variables, semaphores, or message queues.
- **Precise timing and delays**: Enables accurate scheduling of actions like motor updates and sensor polling.

In our RoboKar project, we used **µC/OS-II**, an RTOS that supports deterministic task scheduling and is well-suited for resource-constrained microcontrollers. The RoboKar runs several concurrent tasks:

- A **Navigation Task** that reads the line sensors and determines movement direction.

- A **Collision Detection Task** that monitors the proximity sensor and triggers avoidance behaviors.

- A **Motor Control Task** that constantly updates motor speeds based on the robot's current state.

The design allows the robot to respond efficiently on more than one input at a time, a feature that adds more responsiveness and stability. Just as an example, when the robot goes on the curved pathway and still senses the obstacle, it is able to stop, reverse and turn all without a hitch in the overall movement process.

Concurrent programming is essential in real-time robotics because it would not result in one of the functions being blocked or delayed by another one. In its absence, systems such as RoboKar would be sluggish, faulty, and incompatible to dynamically changing areas like the RoboKar competition track.

3.2.2 RoboKar (Introduce Robot Programming)

RoboKar is an educational robot, a mobile competition robot that is capable of autonomously navigating a straight line on a track and performing several tasks at real-time. The project offers a concrete solution to embedded system design, concurrent programming and real-time task scheduling with the use of an RTOS.

On the basic level, a microcontroller-based platform (Atmel ATmega328P) and the C language are laid for RoboKar. It employs a mix of actuators and sensors in order to connect to or execute in the environment:

- **Infrared line sensors** to follow a predefined black path on the ground.
- **Proximity sensors** to detect nearby obstacles.
- **Light sensors** to respond to bright light triggers (used in L1 task).
- **DC motors** to control movement and steering.
- **Buzzer and LEDs** to provide audible and visual feedback.

Programming the RoboKar involves more than writing linear sequences of instructions. The robot is designed with a real-time operating system that has the capability of multitasking i.e. µC/OS-II. Since the robot has to perform many tasks simultaneously (such as: drive and at the same time look out for obstacles in the way), the operating system needs to be capable of handling many tasks at the same time. Each task receives a certain priority and operates in its own, and the results enable the deterministic regulation of time-sensitive behavior.

The primary tasks in RoboKar's program include:

- **Navigation Task**: Uses the line sensor to determine the robot's position on the track and adjusts motor speeds accordingly to stay on the line.

- **Collision Detection Task**: Monitors the proximity sensor and initiates a stop-reverse-turn sequence if an obstacle is detected.

- **Motor Control Task**: Applies the current left and right motor speed values to ensure continuous, smooth movement.

- **Startup Task**: Initializes the RTOS, creates tasks, and monitors system indicators (e.g., LED toggling).

Each task operates on a shared robot state structure (`myrobot`) that contains variables such as motor speeds and obstacle flags. The use of a shared data structure enables coordinated decision-making across tasks while still allowing each function to execute independently.

By separating robot behaviors into modular tasks and leveraging real-time scheduling, the RoboKar achieves a balance of **simplicity, responsiveness, and modularity**. This design also aligns with industry practices in embedded robotics, where real-time constraints and task-based systems are standard.

This programming method is not only built to be efficient; it is highly customizable. As an example, new functionality (blinking of LEDs in presence of light, or selection of speeds in case of curvy tracks) can be acquired as the new tasks without interfering with the already available functionality. The proposed modular architecture is the basis of the two following implementation and testing phases of the RoboKar development lifecycle.

3.3 Robokar Concurrent Design And Implementation (Phase 1 – GI1)

The RoboKar's software is built on a **concurrent programming model** using **μC/OS-II**, where each core functionality of the robot is implemented as a separate task. This modular and real-time architecture ensures high responsiveness to external events, which is critical for autonomous navigation in dynamic environments such as the RoboKar track.

3.3.1 RoboKar Concurrent Requirements and Design

The RoboKar must perform multiple operations simultaneously:

- **Follow a black line path**
- **Avoid obstacles when detected**
- **React to light stimulus**
- **Maintain motor speed control continuously**

These tasks require **parallel execution** with minimal delay. Based on these requirements, the following RTOS-based task design was created:

Task Name	Purpose	Priority
TaskStart	Initializes the system and creates other tasks	1 (lowest)
CheckCollision	Monitors proximity sensor for obstacles	2
CntrlMotors	Applies motor speed continuously based on robot state	3
Navig	Reads line and light sensors to control navigation	4 (highest)

Each task has its own **dedicated stack** so the time-sensitive processes such as navigation take precedence, and other non-critical tasks such as updating motor speeds are moved to the back. **Preemptive multitasking** is used to schedule the tasks which ensures real-time performance.

A **shared data structure**, `myrobot`, is used to represent the current state of the robot, including:

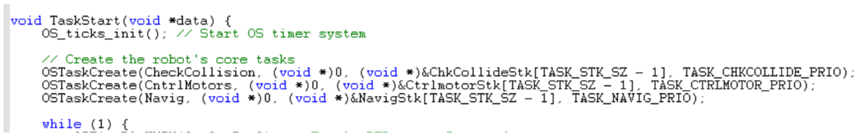
- `rspeed`: Right motor speed

- `lspeed`: Left motor speed
- `obstacle`: A flag indicating whether an obstacle is present

This structure allows all tasks to update or read robot state in a synchronized way.

3.3.2 RoboKar Implementation Using μC/OS-II RTOS

The implementation begins with hardware setup (`robo_Setup()`) and RTOS initialization (`OSInit()`). The `TaskStart()` function is then created as the first task and is responsible for spawning the remaining three core tasks:



```
void TaskStart(void *data) {
    OS_ticks_init(); // Start OS timer system

    // Create the robot's core tasks
    OSTaskCreate(CheckCollision, (void *)0, (void *)&ChkCollideStk[TASK_STK_SZ - 1], TASK_CHKCOLLIDE_PRIO);
    OSTaskCreate(CntrlMotors, (void *)0, (void *)&CtrlmotorStk[TASK_STK_SZ - 1], TASK_CTRLMOTOR_PRIO);
    OSTaskCreate(Navig, (void *)0, (void *)&NavigStk[TASK_STK_SZ - 1], TASK_NAVIG_PRIO);

    while (1) {
```

Each task then enters its own infinite loop (`for(;;)`) where it performs its respective operations:

- `Navig`: Reads `robo_lineSensor()` and `robo_lightSensor()` to control speed and direction.
- `CheckCollision`: Uses `robo_proxSensor()` to detect obstacles and perform avoidance maneuvers (stop, reverse, turn).
- `CntrlMotors`: Continuously applies `myrobot.lspeed` and `rspeed` to the motors using

`robo_motorSpeed()`.

To ensure time-synchronized behavior, each task uses `OSTimeDlyHMSM(...)` to define task intervals:

- `Navig`: 5ms sensor check rate
- `CheckCollision`: 100ms obstacle check
- `CntrlMotors`: 250ms motor speed refresh

This configuration ensures real-time responsiveness and stability during robot operation.

3.4 RoboKar Testing and Performance Analysis (Phase 2)

Real-time behavior was observed and evaluated in the case of RoboKar, according to various setTimeout conditions (task period, priority) which is a vital characteristic in applications. Moreover, the impact of the task scheduling on the behavior of RoboKar in the RoboKar Game was also examined.

We conducted a number of experiments by:

- Altering task periods (e.g., increasing/decreasing sensor check intervals)
- Reordering the priorities of tasks (e.g., decreasing collision task or promoting motor task)
- Seeing the movements of the robot concerning:
  - Delayed response to lines, lights, or obstructions
  - Accomplishment of checkpoints (A–F)
  - Movement consistency

Timing Analysis Table

Task	Period (ms)	Priority	Response Time	Observation Summary
Navig	5	4	~1 ms	Smooth tracking, no deviation
CheckCollision	100	2	~2 ms	Detected obstacle in time, reversed early
CntrlMotors	250	3	~1 ms	Motor updated continuously, no stuttering
CheckCollision (Priority 1)	100	1	~2 ms	Obstacle was not avoided on time, collision occurred
Navig (10 ms)	10	4	~2–3 ms	Slight delay in turns, minor path deviation

- Real-time responsiveness depends on accurate priority setting and precise timing.
- Mission failure may result from improper task scheduling, such as decreasing the priority of collision detection.

- With no deadlines missed, the default setup (Navig = 5 ms, CheckCollision = 100 ms, CntrlMotors = 250 ms) operated at its optimum performance.

3.5 Discussion (Phase 1 – GI 2)

3.5.1 Comparison of Mobile Robot Development Approach

Feature	**RoboKar**	**Research Paper (Manikandan et al., 2023)**
Platform	ATmega328P + IR, Proximity, Light	Raspberry Pi + GPS, IMU, Hall Effect
Steering Mechanism	DC Motors	Linear Actuator
Navigation Logic	Line-following logic	Curve-aware Pure Pursuit (C-PP)
Software Architecture	µC/OS-II RTOS	Not explicitly RTOS-based
Timing Analysis	Manual + task scheduling	RMSE, angle error, simulation & field testing

- The article in a reference does not have noticeable scheduling methods, instead, it uses complex navigation algorithms, however, the RoboKar demonstrates the effective task scheduling in real-time, which makes use of RTOS..

- RoboKar prioritizes simplicity and predictability, while the design of the paper offers greater precision by utilizing advanced sensing and actuation.

3.5.2 Software Re-development Approach and Improvement

1. Introduce RTOS (for example, μC/OS-II)
    - Structure tasks for actuation, steering control, and sensor reading.
2. Implement Modular, Preemptive Task Scheduling
    - Separate tasks for obstacle detection, navigation, and actuators with fixed time.
3. Use Structured Shared Memory Communication
    - Use a global structure, such as myrobot, to control communication between tasks.
4. Apply Adaptive Timing Based on Curvature
    - Adjust task priorities and frequencies dynamically to the curvature or terrain at any given time.
5. Integrate Line Tracking and Sensor Fusion
    - For better accuracy, combine the paper's IMU/GPS system with RoboKar's reactive model.

3.6 Conclusion

Concurrent task design with μC/OS-II in an embedded real-time system. The modularity of the tasks, e.g. the motor control, collision detection, and navigation provided consistency of responsive performance to RoboKar.

Accurate task periods and priorities are crucial for real-time behavior, confirmed by timing analysis. In literature, RoboKar's abilities in predictability and its potential for increased precision and flexibility were emphasized through comparison with an advanced mobile robot system.

Future improvements will include adaptive task control, sensor fusion, and advanced navigation algorithms such as Curve-aware Pure Pursuit.

ACKNOWLEDGEMENT

The RoboKar experience has not just meant code and circuits, but it has been a learning experiment at the level of communication, team commitment, and developing creative solutions. We thank our lecturer, Prof. Ts. Dr. Dayang Norhayati Bte. Abang Jawawi with all our heart whose sheer patience and technical expertise has guided us through real time systems deadlock in logic and through sensor bugs and tunnel-vision. To Universiti Teknologi Malaysia (UTM), we would like to thank you for providing us an environment where innovation can be inculcated by ideas. and to us, Group CodeSync - Khoo Ann Hong, Lai Zhi Yuan and Nur Amirah Athirah Binti Mohd Nazli, and Nur Farra Aqila Binti Roslee, thanks to the sleepless nights, the lively arguments and the belief that RoboKar could actually move more than a line on the floor. This project reflects our collective evolution as engineers and collaborators.

REFERENCES

Liu, J. W. S. 2000. *Real-Time Systems*. Prentice Hall.

Burns, A., & Wellings, A. 2009. *Real-Time Systems and Programming Languages*. Addison-Wesley.

Manikandan, N. S., & Kaliyaperumal, G. 2023. *Hardware and Software Design for Mobile Robot's Navigation Over On-Road and Off-Road Curvature Paths. IEEE Access, 11, 89625–89643. doi:10.1109/ACCESS.2023.3305678*