Sprint 3 Week 1 - Command Interface Development Documentation

Overview

This sprint focused on extending the **DeskBuddy robot controller** with a **simple command interface**, allowing it to interpret local text-based or keyboard commands such as *“add task”*, *“list tasks”*, or *movement commands*. The goal was to bridge low-level control (wheel motors, LEDs, etc.) with higher-level task automation through structured command parsing.

Objectives

#	Objective	Status
1	Extend robot controller to handle text commands	✅ Completed
2	Map commands to robot actions (move, blink, speak, etc.)	✅ Completed
3	Implement simple task system (“add task”, “list tasks”)	✅ Completed
4	Test command sequence handling	✅ Completed

Key Features Implemented

1. Command Interface

A lightweight **command processor** was added to interpret text commands entered via keyboard or local input.

Example Commands:

Command	Description
`forward`	Moves robot forward
`backward`	Moves robot backward
`left`	Turns robot left
`right`	Turns robot right
`stop`	Stops all movement
`blink`	Blinks the robot’s LED eyes
`speak <message>`	Makes the robot say a message
`add task <description>`	Adds a new task to robot’s internal list
`list tasks`	Prints all stored tasks

---

2. Command Mapping Architecture

```
def process_command(self, command):
    cmd = command.lower()

    if cmd.startswith("add task"):
        task = cmd.replace("add task", "").strip()
        self.tasks.append(task)
        print(f"Task added: {task}")

    elif cmd == "list tasks":
        for i, t in enumerate(self.tasks, 1):
            print(f"{i}. {t}")

    elif cmd == "forward":
        self.move_forward_c()

    elif cmd == "backward":
```

```
            self.move_backward_c()

        elif cmd == "left":
            self.turn_left_c()

        elif cmd == "right":
            self.turn_right_c()

        elif cmd == "blink":
            self.run_async(self.blink_lights)

        elif cmd.startswith("speak"):
            msg = cmd.replace("speak", "").strip()
            self.run_async(lambda: self.speak(msg))

        elif cmd == "stop":
            self.stop_c()

        else:
            print(f"Unknown command: {cmd}")
```

Key Idea:
The controller maps each recognized command string to an action function (like `move_forward_c()` or `blink_lights()`), similar to a chatbot but for robot actions.

---

3. Task System

A simple **task list** was introduced using Python lists.

Example Interaction:

```
 Command: add task Deliver message to lab
Output: Task added: Deliver message to lab

Command: add task Return to base
Output: Task added: Return to base

Command: list tasks
```

```
Output:
1. Deliver message to lab
2. Return to base
```

This is the early foundation for a full **task management module** in future sprints (to schedule, prioritize, and execute tasks autonomously).

---

4. Command Input & Testing

Commands were tested via:

- **Keyboard input loop**
- **Postman requests** (simulating API command messages)
- **Sequential testing** to ensure commands run safely one after another

Example Test Sequence:

```
 speak Hello DeskBuddy active
forward
blink
add task Patrol perimeter
list tasks
stop
```

Expected Behavior: Robot speaks, moves forward, blinks lights, logs the task, lists all tasks, and stops.

Debugging and Fixes

Issue	Cause	Solution
“Robot not initialized” error	Command sent before `Robot()` was created	Added safe initialization check and error handling
Commands freezing Webots	Running blocking functions in main thread	Moved actions to threads using `run_async()`

Issue	Cause	Solution
Inconsistent command parsing	Missing `strip()` on input	Sanitized input to remove whitespace before parsing
Conflicts between tasks and motion threads	No lock synchronization	Reused `action_lock` for critical actions

---

Technical Takeaways

1. **Command-driven control** bridges manual input and future AI autonomy.
2. **Threaded architecture** prevents UI or motion blocking.
3. **Synchronous vs Asynchronous Commands** must be carefully separated.
4. **Task lists** enable future scheduling and persistence.
5. **Integration with APIs** (via Postman or local server) is now possible.

Code Files Updated

File	Purpose
`desk_buddy_controller.py`	Added command parser, task list, and command mapping
`testworld.wbt`	No structural change this sprint
`command_test.py`	Local command testing and debugging utility

Problems Faced

- Difficulty ensuring thread safety between movement and speech commands.
- Parsing errors when extra spaces were present in commands.
- Postman request errors when the robot was not yet initialized (resolved by adding `is_initialized` flag).

- Confusion between Webots real-time steps and local command delays — mitigated using consistent `timestep` management.

Sprint Outcome

- ✅ Robot can interpret and act on textual or keyboard-based commands
- ✅ Commands mapped to both motion and expressive actions
- ✅ Basic task management implemented
- ✅ Ready for integration with external API command sources

Next Sprint Goals (Sprint 3 Week 2)

1. Connect command interface to **API endpoint** (receive JSON commands)
2. Add **response messages** from robot to API (confirmation/status)
3. Implement **task persistence** (save/load task list)
4. Add **error feedback system** (invalid command handler)
5. Begin foundation for **voice command parsing**