Sprint 3 Week 2 - Webots Robot Development Documentation

Overview

This sprint focused on building a 4-wheeled robot controller in Webots with keyboard input, understanding the physics simulation, and debugging movement issues.

Key Concepts Covered

1. Physics Fundamentals

Newton's Laws Applied to Robotics:

- F = ma (Force equals mass times acceleration)
- Velocity increases when forces are applied
- Friction slows movement
- Gravity affects vertical position
- Collision detection prevents objects from passing through each other

Implementation in Webots:

- Webots automatically calculates physics each timestep
- Developers apply forces/velocities to motors
- The engine handles collision responses
- `basicTimeStep` controls simulation frequency (16ms in our case = 62.5 updates/second)

2. VRML World Files (.wbt)

Structure:

```
#VRML_SIM R2025a utf8  ← MUST be first line
WorldInfo { ... }       ← Global settings
Viewpoint { ... }       ← Camera position
Robot { ... }           ← Controllable robot
Solid { ... }           ← Static/dynamic objects
```

Key Components:

- `DEF name` - Defines/names an object for identification and reuse
- `USE name` - References a previously defined object
- `HingeJoint` - Rotating joint (for wheels, head tilt, etc.)
- `RotationalMotor` - Actuator that drives a joint
- `boundingObject` - Physics collision shape (separate from visual)
- `physics Physics {}` - Enables physics simulation

Common Mistakes:

- Wrong VRML header (must be `#VRML_SIM R2025a utf8`)
- Missing `boundingObject` or `physics` blocks
- Motor names not matching Python `getDevice()` calls
- Swapped wheel anchor positions causing incorrect turning

3. Robot Controller Architecture

Python Webots API:

```
from controller import Robot, Keyboard

robot = Robot()
timestep = int(robot.getBasicTimeStep())
motor = robot.getDevice("motor_name")
motor.setVelocity(speed)

while robot.step(timestep) != -1:
    # Main control loop
    pass
```

Key Rules:

- Only the main thread calls `robot.step()`
- Threaded actions use `time.sleep()` instead of `robot.step()`
- Thread safety requires locks (`threading.Lock()`)
- `getDevice()` names must match `.wbt` motor names exactly

4. Keyboard Input

Basic Implementation:

```
keyboard = robot.getKeyboard()
keyboard.enable(timestep)

key = keyboard.getKey()
if key == ord('W'):
    # Do something
```

Key Points:

- Enable keyboard with sampling period
- `getKey()` returns one key per step
- Use `ord('character')` to get key codes
- Special keys: `keyboard.KEY_UP` , `keyboard.KEY_DOWN` , etc.

5. 4-Wheel Robot Movement

Wheel Positioning (X, Y, Z):

- Front left: (-0.1, -0.15, 0.1)
- Front right: (0.1, -0.15, 0.1)
- Rear left: (-0.1, -0.15, -0.1) ← CRITICAL: Not swapped
- Rear right: (0.1, -0.15, -0.1) ← CRITICAL: Not swapped

Movement Commands:

- **Forward**: All 4 wheels same velocity
- **Backward**: All 4 wheels negative velocity
- **Turn Left**: Left side backward, right side forward
- **Turn Right**: Left side forward, right side backward

Bug Fixed:

Original had rear wheel X positions swapped, causing turn conflicts.

6. Threading Architecture

Pattern Used:

```
class Robot:
    def __init__(self):
        self.action_lock = threading.Lock()
        self.active_threads = []

    def run_async(self, func):
        def wrapper():
            try:
                func()
            finally:
                # Cleanup

        thread = threading.Thread(target=wrapper)
        self.active_threads.append(thread)
        thread.start()
```

Benefits:

- Simultaneous actions (speak + move + wave)
- Non-blocking operations
- Thread-safe device access with locks

DeskBuddy Robot Specifications

Hardware Devices:

- 4 wheel motors (front left, front right, rear left, rear right)
- 1 tilt motor (head)
- 2 LEDs (eyes)
- 1 speaker

Dimensions:

- Body: 0.25m × 0.3m × 0.25m
- Wheels: 0.04m height, 0.05m radius
- Mass: Default (determined by density)

Control Modes:

- Direct control: `setVelocity()` for continuous movement
- Threaded actions: Time-based movements with `time.sleep()`

Code Files Delivered

1. World File (testworld.wbt)

- Defines robot structure with 4 wheels
- Sets up head tilt motor
- Configures LED eyes and speaker
- Includes physics for all objects

2. Controller (desk_buddy_controller.py)

Direct Movement Methods:

- `move_forward_c()` - All wheels forward
- `move_backward_c()` - All wheels backward
- `turn_left_c()` - Rotate left
- `turn_right_c()` - Rotate right
- `stop_c()` - All wheels stop

Threaded Actions:

- `speak(message)` - Speak with LED animation
- `wave()` - Tilt head left/right
- `blink_lights()` - Flash LEDs
- `move_forward(duration)` - Move for X seconds
- `move_backward(duration)` - Move back for X seconds
- `turn(direction, duration)` - Turn for X seconds

- `dance_sequence()` - Coordinated LED + head movement
- `patrol_mode()` - Continuous patrol pattern

Keyboard Controls:

- W: Wave
- B: Blink LEDs
- H: Say Hello
- P: Patrol Mode
- D: Dance
- F/R/L/G: Move forward/backward/turn left/turn right
- Space: Stop movement
- Y: All actions simultaneously
- T: Turn left & speak together
- S: Stop all actions
- Q: Quit simulation

Debugging Process

Issues Encountered:

1. **VRML Header Error**
   - Problem: Using old VRML V2.0 format
   - Solution: Changed to `#VRML_SIM R2025a utf8`
2. **Turning Not Working**
   - Problem: Rear wheel anchor X positions were swapped
   - Solution: Corrected LEFT_REAR to -0.1, RIGHT_REAR to 0.1
3. **Unique Name Warnings**
   - Problem: Multiple Solid nodes named “solid”
   - Solution: Renamed to descriptive names (head_solid, left_wheel_solid, etc.)
4. **Movement Functions Missing Rear Wheels**
   - Problem: Only front wheels controlled in some methods
   - Solution: Added rear wheel setVelocity() calls to all movement functions

---

Technical Takeaways

1. **Coordinate System Matters**: Wheel positions directly affect movement quality
2. **Thread Safety**: Lock-based synchronization prevents device conflicts
3. **Physics Simulation**: Webots handles most calculations automatically
4. **Name Consistency**: Motor names in `.wbt` must match `getDevice()` calls
5. **Separation of Concerns**: Visual shapes separate from collision objects

---

Files Status

✅ World file: Complete and tested
✅ Controller: Complete with all features
✅ Keyboard input: Working
✅ 4-wheel movement: Working
✅ Threading: Working
✅ All actions: Tested

Next Sprint Goals

1. Add sensor input (distance, light, IMU)
2. Implement autonomous navigation
3. Add obstacle avoidance logic
4. Create state machine for complex behaviors
5. Add data logging for debugging

References

- Webots Documentation: https://cyberbotics.com/doc
- Python Controller API: Threading, Robot, Motor classes
- Physics Engine: VRML/SIM collision and dynamics