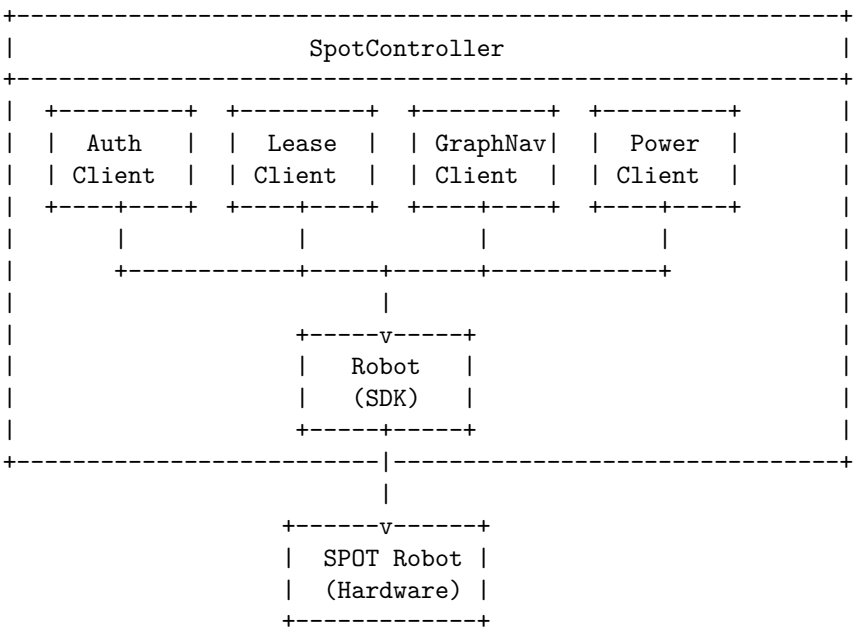
SPOT Controller Architecture

This document explains how the SPOT robot controller works at a conceptual level.

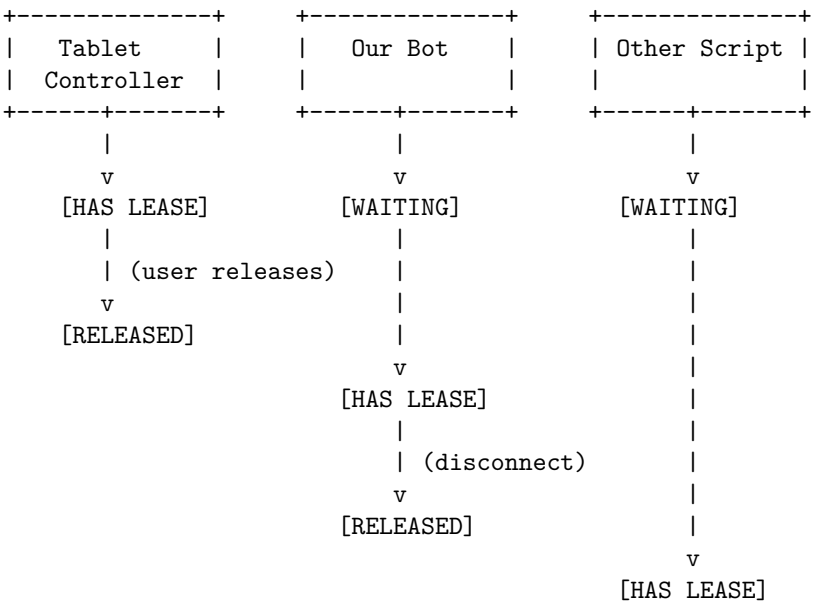
Overview

The `SpotController` manages all communication with the Boston Dynamics SPOT robot. It handles authentication, control permissions, map loading, and navigation.



The Lease System

Why leases? SPOT can only be controlled by one client at a time. The lease system prevents conflicting commands.



LeaseKeepAlive

The lease must be periodically renewed or it expires. `LeaseKeepAlive` handles this automatically:

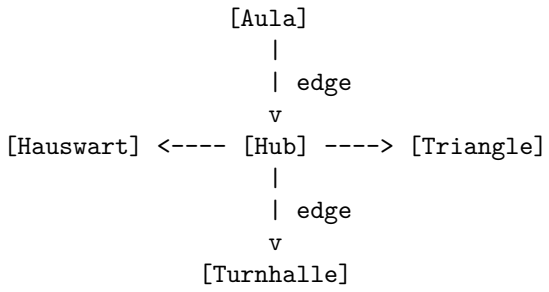
```
self.lease_keepalive = LeaseKeepAlive(
    self.lease_client,
    must_acquire=True,      # Fail if can't get lease
    return_at_exit=True     # Release when program exits
)
```

Common issue: If the tablet has the lease, our bot can't connect. Solution: Release control on tablet first.

GraphNav: How Navigation Works

SPOT uses **GraphNav** for autonomous navigation. Think of it as a pre-recorded "map" of waypoints and paths.

The Graph Structure



- **Waypoints**: Named locations the robot knows
- **Edges**: Paths connecting waypoints
- **Snapshots**: Visual data for localization

Navigation Process

```
1. Load Graph from Disk
   +-- Read waypoints, edges, visual data

2. Upload to Robot
   +-- Robot now knows the map

3. Localize Robot
   +-- Robot finds itself using visual markers (fiducials)

4. Navigate to Waypoint
   +-- Robot plans path and walks there
```

Waypoint Identification

Waypoints have complex IDs like `aula-vast-abc123-xyz789`. We use shortcuts:

Short Code	Full ID Pattern	Location
`al`	`aula-lobby-...`	Aula (assembly hall)
`tv`	`triangle-vast-...`	Triangle
`oh`	`office-hauswart-...`	Hauswart
`cw`	`corridor-west-...`	Turnhalle

The `find_unique_waypoint_id()` function resolves these:

```
"aula"  ->  looks up annotation name  ->  full waypoint ID
"al"    ->  searches graph for match  ->  full waypoint ID
```

Connection Sequence

```
await controller.connect(status_callback)
```

This triggers a 4-step sequence:

```
Step 1: Authenticate
|-- Create SDK instance
|-- Connect to robot IP
|-- Login with credentials
+-- Sync clocks

Step 2: Acquire Lease
|-- Request exclusive control
|-- Start keepalive heartbeat
+-- (Fails if tablet has lease)

Step 3: Upload Map
|-- Load graph from disk
|-- Load waypoint snapshots
|-- Load edge snapshots
+-- Upload all to robot

Step 4: Localize
|-- Robot looks for fiducial markers
|-- Matches to known positions
+-- Robot now knows where it is
```

Power State Management

SPOT's motors can be on (standing) or off (sitting). We track this:

```
self._powered_on = False           # Current state
self._started_powered_on = False   # State when we connected
```

Why two variables?

If the robot was already standing when we connected, we shouldn't make it sit down after navigation. We restore the original state:

```
# Before navigation
if not powered_on:
    power_on()  # Make robot stand

# After navigation
if we_powered_it_on and not started_powered_on:
    power_off()  # Put robot back to sitting
```

The Heartbeat Pattern

Navigation takes time (30+ seconds). We send periodic status updates:

```
Time     Status Callback              Robot State
-------------------------------------------
0s       "Navigating to Aula..."      Starting
3s       "Navigating to Aula (3s)"    Walking
6s       "Navigating to Aula (6s)"    Walking
9s       "Navigating to Aula (9s)"    Walking
12s      "Arrived at Aula!"           Reached goal
```

This keeps users informed and confirms the bot isn't frozen.

Async/Thread Boundary

The SPOT SDK uses **blocking** calls, but our bot is **async**. We bridge this with `asyncio.to_thread()`:

```
# This would block the entire bot:
self.robot.authenticate(user, pass)  # BAD

# This runs in a thread pool, doesn't block:
await asyncio.to_thread(
    self.robot.authenticate, user, pass
)  # GOOD
```

Rule: All SPOT SDK calls go through `asyncio.to_thread()`.

Navigation Status Codes

The robot reports its navigation progress:

Status	Meaning	Action
`STATUS_REACHED_GOAL`	Success!	Report arrival
`STATUS_LOST`	Can't find itself	Report error
`STATUS_STUCK`	Can't move	Report error
`STATUS_ROBOT_IMPAIRED`	Hardware issue	Report error
(other)	Still navigating	Keep waiting

Error Handling Strategy

```
try:
    await controller.connect(callback)
except ResourceAlreadyClaimedError:
    # Someone else has the lease (probably tablet)
    # Tell user to check tablet
except Exception as e:
    # Log full error for debugging
    logger.exception("Connection failed")
    # Show user-friendly message
    await callback(f"Connection failed: {e}")
```

Key Files

File	Purpose
`src/spot/spot_controller.py`	Main controller class
`src/spot/__init__.py`	Exports `SpotController`, `WAYPOINTS`
`maps/map_catacombs_01/`	Pre-recorded navigation map

Map Directory Structure

```
maps/map_catacombs_01/
|-- graph                    # Navigation graph (protobuf)
|-- waypoint_snapshots/      # Visual data for each waypoint
|   |-- snapshot_abc123
|   |-- snapshot_def456
|   +-- ...
+-- edge_snapshots/          # Visual data for paths
    |-- edge_snap_001
    |-- edge_snap_002
    +-- ...
```

Reliability Features

Graceful Shutdown

When the bot stops (Ctrl+C or SIGTERM), it automatically releases the lease:

```
Bot running -> Ctrl+C pressed
                    |
                    v
            post_shutdown() called
                    |
                    v
            spot_controller.disconnect()
                    |
                    v
            Lease released cleanly
                    |
                    v
            Next bot start -> Can acquire lease!
```

Force Connect

If the lease is stuck (from a crashed bot), use `/forceconnect`:

```
/forceconnect -> Takes lease from ANY client
                    |
                    v
            Previous owner disconnected
                    |
                    v
            Bot now has control
```

Status Monitoring

Use `/status` to check: - Connection state - Motor power (sitting/standing) - Battery percentage - E-stop status - Current lease owner

Common Issues & Solutions

Issue	Cause	Solution
"Lease already claimed"	Tablet or crashed bot has lease	Use `/forceconnect`
"Failed to localize"	No fiducial visible	Move robot to see a marker
"Robot got stuck"	Obstacle in path	Clear the path
"Connection refused"	Wrong IP or robot off	Check IP, power on robot
Can't reconnect after restart	Previous bot didn't release lease	Use `/forceconnect`

Further Reading

- Boston Dynamics SDK Docs
- GraphNav Documentation
- `spot-sdk/` submodule contains example code