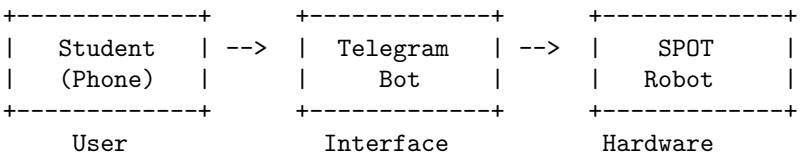
Telegram Bot Architecture

This document explains how the Telegram bot component works at a conceptual level.

Overview

The Telegram bot serves as the **user interface** for the Gipfeli delivery system. Students interact with SPOT through Telegram messages and button presses, never directly with the robot.



How Telegram Bots Work

Telegram bots are programs that receive messages from users and respond. The key concepts:

1. **Bot Token**: A secret key from BotFather that identifies your bot
2. **Updates**: Messages/events from users (commands, button clicks)
3. **Handlers**: Functions that process specific types of updates
4. **Polling**: Continuously asking Telegram "any new messages?"

Command Flow

When a user sends `/goto`:

```
1. User taps "/goto" in Telegram
         |
         v
2. Telegram servers receive the message
         |
         v
3. Our bot (polling) fetches the update
         |
         v
4. CommandHandler("goto", goto) matches it
         |
         v
5. goto() function runs, sends button menu
         |
         v
6. User sees location buttons in chat
```

Handlers Explained

Command Handlers

Respond to `/commands`:

Command	Handler	Purpose
`/start`	`start()`	Greet new users
`/help`	`help_command()`	Show available commands
`/connect`	`connect_spot()`	Connect to SPOT robot
`/disconnect`	`disconnect_spot()`	Release lease and disconnect
`/forceconnect`	`forceconnect_spot()`	Force take lease from any client
`/status`	`status_spot()`	Show robot status (battery, motors, lease)
`/goto`	`goto()`	Show location buttons

Callback Handler

Responds to **inline button presses**:

```
# When user taps "Aula" button:
callback_data = "goto_aula"
            |
            v
CallbackQueryHandler(goto_callback, pattern="^goto_")
            |
            v
goto_callback() extracts "aula" and navigates
```

Async/Await Pattern

All handlers are `async` functions. This is important because:

1. **Non-blocking**: Bot can handle multiple users simultaneously
2. **Status updates**: Long operations (navigation) can send progress messages
3. **Telegram requirement**: The library requires async handlers

```
async def connect_spot(update, context):
    # This doesn't block other users
    await update.message.reply_text("Connecting...")

    # Robot connection happens in background thread
    success = await spot_controller.connect(send_status)

    # Bot remains responsive during connection
```

Status Callbacks

For long-running operations, we pass a callback function:

```
async def send_status(msg: str):
    await update.message.reply_text(msg)

# SpotController calls this during connection:
# "Connecting to SPOT..."
# "Authenticated with SPOT"
# "Acquiring lease..."
# "Lease acquired"
# etc.
```

This keeps users informed during operations that take several seconds.

Global State

The bot maintains one `SpotController` instance globally:

```
spot_controller: Optional[SpotController] = None
```

Why global? - Only one robot connection needed - All handlers need access to the same controller - Telegram's async model is single-threaded, so this is safe

Trade-off: Not ideal for testing (we use `patch` to mock it).

Lifecycle Hooks

The bot uses lifecycle hooks to manage SPOT connection automatically.

Auto-Connect on Startup

```
async def post_init(application):
    # Called once after bot starts
    spot_controller = SpotController(hostname, map_path)
    await spot_controller.connect(log_status)
```

If auto-connect fails, users can manually connect with `/connect`.

Graceful Shutdown

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

```
async def post_shutdown(application):
    # Called when bot is shutting down
    if spot_controller and spot_controller.is_connected:
        await spot_controller.disconnect()
```

Why this matters: Without graceful shutdown, the lease stays "claimed" and you can't reconnect without using `/forceconnect` or the tablet.

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

Message Flow Diagram

```
User Action          Bot Response              Robot Action
-----------------------------------------------------------
/start          ->   "Hi [name]!"
/help           ->   Command list
/connect        ->   "Connecting..."      ->   Authenticate
                     "Authenticated"      <-   SDK connected
                     "Lease acquired"     <-   Lease obtained
                     "Map uploaded"       <-   Graph loaded
                     "Localized!"         <-   Position found
                     "SPOT ready!"
/status         ->   Connection, battery,
                     motors, lease info
/goto           ->   [Button menu]
[Tap "Aula"]    ->   "Navigating..."      ->   Start moving
                     "Navigating (3s)"    <-   Heartbeat
                     "Navigating (6s)"    <-   Heartbeat
                     "Arrived at Aula!"   <-   Goal reached
/disconnect     ->   "Disconnecting..."   ->   Release lease
                     "Disconnected"
/forceconnect   ->   "Force connecting..."->   Take lease
                     "Lease forcefully       from any client
                      acquired"
```

Error Handling

The bot handles errors gracefully:

```
try:
    await query.edit_message_text(msg)
except BadRequest:
    # Message was deleted or already edited
    logger.debug("Could not update message")
except Exception as e:
    # Unexpected error
    logger.warning(f"Error: {e}")
```

Users see friendly error messages, not stack traces.

Key Files

File	Purpose
`src/telegram/bot.py`	Main bot code
`src/logging_config.py`	Logging setup
`.env`	Bot token (secret!)

Troubleshooting

Problem	Solution
Can't connect after restart	Use `/forceconnect` to take lease
"Lease already claimed"	Someone else has control - use `/forceconnect`
Bot unresponsive	Check `logs/telegram.log` for errors
Unknown connection state	Use `/status` to check robot state

Further Reading

- python-telegram-bot docs
- Telegram Bot API