Turret System Client–Server Application Documentation

GelBlaster - Wingman

February 2025

Contents

1 Introduction

1.1 Overview

This documentation covers the client–server turret system application designed to provide remote control, camera feed handling, object detection, and hardware control (including servos and relay mechanisms) via MQTT communication. The system integrates software components with hardware (cameras, Dynamixel servos, GPIO, etc.) and provides both manual and automatic operational modes.

1.2 Purpose and Scope

- **Purpose:** To document the design, architecture, installation, configuration, and usage of the turret system application. This documentation serves as a technical reference for developers, integrators, and maintainers.
- **Scope:** The document covers both the client and server code, explains the MQTT topics used for communication, details hardware requirements, and provides troubleshooting and future enhancement guidelines.

1.3 Target Audience

- **Robotics Enthusiasts:** Makers and hobbyists interested in building automated turret systems or similar robotic projects
- **Software Developers:** Engineers working on computer vision, robotics control systems, or MQTT-based applications
- **Hardware Integrators:** Those working with Dynamixel servos, cameras, and GPIO systems
- **Gelblaster Community:** Users looking to enhance their Gelblaster XL with automated targeting capabilities
- **Venue Operators:** Gelblaster arena and entertainment venue operators interested in adding automated turret systems to their facilities
- **Academic/Research:** Students and researchers in robotics, computer vision, or human-robot interaction

2 System Architecture

2.1 High-Level Overview

The turret system is composed of two major parts:

Client Application Provides a graphical user interface (GUI) using OpenCV, processes incoming camera feed images, handles user inputs (keyboard and mouse), and sends control commands via MQTT.

Server Application Interfaces with hardware components (camera, servos, relay), processes sensor data, detects objects (e.g., green objects), controls servo movements, and triggers the relay. It publishes the camera feed and status updates over MQTT.

2.2 Client and Server Roles

Client:

– Receives live video feed from the server.

– Provides a control panel with manual override options (e.g., toggle auto/manual mode, adjust speed).

– Processes keyboard and mouse events to send directional commands.

– Publishes commands (e.g., pan/tilt adjustments, trigger commands) to MQTT topics.

Server:

– Captures video frames using a connected camera (or an OAK-D device).

– Processes frames for object detection (e.g., identifying green objects).

– Reads and writes servo positions via Dynamixel controllers.

– Controls hardware relays (with GPIO) and publishes sensor/servo status.

– Subscribes to MQTT topics to receive command updates from the client.

2.3 MQTT Communication and Topics

The system uses the Paho MQTT library for messaging. Key topics include:

– **Client-to-Server Topics:**

  * `dpad/commands`: Contains directional commands, auto-mode toggles, and speed settings.
  * `relay/command`: Commands for relay activation/deactivation.
  * `server/sensitivity`: Commands to adjust sensitivity settings.

– **Server-to-Client Topics:**

  * `camera/feed`: Encoded JPEG images from the camera.
  * `camera/bbox`: Bounding box data for detected objects.
  * `servo/status`: Current servo positions and status updates.
  * `server/tof`: Time-of-flight sensor data (if applicable).

2.4 Hardware and Software Dependencies

Hardware:

- Camera (webcam or OAK-D device)
- Dynamixel Servos (Pan and Tilt)
- Relay (controlled via GPIO, e.g., on a Linux-based system)
- (Optional) Input devices (gamepad, keyboard)

Software:

- Python 3.x
- OpenCV
- NumPy
- Paho MQTT
- Pygame
- Platform-specific libraries (e.g., Jetson.GPIO, depthai, dynamixel_sdk)
- Additional dependencies as required by the hardware (e.g., serial, netifaces)

3 Software Components

3.1 Client Application

- **User Interface:** An OpenCV window displays the camera feed alongside a control panel. The control panel contains an Auto/Manual mode toggle button, a speed slider with touch-friendly controls, and real-time MQTT connection status.
- **Camera Feed Handling:** The client subscribes to the `camera/feed` topic, decodes JPEG images, and overlays additional UI elements (such as crosshairs and bounding boxes).
- **User Input Handling:** Keyboard (e.g., arrow keys, spacebar, ESC) and mouse events are captured. These inputs are translated into MQTT commands for panning, tilting, and firing the relay.
- **MQTT Integration:** The client connects to the MQTT broker, subscribes to several topics, and publishes control messages. Callback functions (e.g., `on_connect`, `on_message`) handle connection events and incoming messages.

3.2 Server Application

– **Hardware Initialization:** On startup, the server configures the camera (using depthai or OpenCV based on the platform), initializes the GPIO for relay control (Linux only), and detects and initializes Dynamixel servos using the Dynamixel SDK.

– **Camera Feed and Processing:** The server continuously captures frames from the camera, processes them (e.g., flips the image, applies object detection algorithms), and publishes the processed frames over MQTT.

– **Servo and Relay Control:** Servo positions are read and written using dedicated functions. Control loops adjust servo positions based on manual commands or automatic corrections derived from object detection. The relay is triggered with debounce logic to prevent rapid switching.

– **Green Object Detection:** The server uses OpenCV to detect green objects within the camera feed. When an object is detected, its bounding box is computed, and if conditions are met (e.g., auto mode enabled and crosshair alignment), the relay is triggered.

– **MQTT Integration:** The server subscribes to command topics and publishes status updates (e.g., `servo/status`, `server/tof`). It processes messages to update sensitivity settings and control servo movements.

4 Installation and Configuration

4.1 System Requirements

– **Operating Systems:**

  * **Linux:** Full hardware integration (GPIO, Dynamixel servos).
  * **macOS:** Limited to simulation or basic webcam usage; certain libraries (e.g., Jetson.GPIO) are not applicable.

– **Hardware Components:**

  * Camera (USB webcam or OAK-D)
  * Dynamixel servos (e.g., U2D2 controller)
  * Relay connected to a GPIO pin (for Linux systems)
  * (Optional) External input devices

4.2 Software Dependencies and Setup

1. **Python 3.x:** Ensure Python 3 is installed.

2. **Required Libraries:** Install the necessary packages using pip:

```
pip install opencv-python numpy paho-mqtt pygame netifaces
```

Additional packages for hardware integration:

- On **Linux:**

```
pip install depthai dynamixel_sdk Jetson.GPIO
```

- On **macOS:** Installation instructions may vary; ensure OpenCV is compiled with AVFoundation support if required.

3. **MQTT Broker:** Set up an MQTT broker (e.g., Mosquitto). Update the broker IP and port in the code as needed.

4.3 Hardware Setup and Wiring

- **Camera:** Connect the camera to the system and verify that it is recognized by the operating system.
- **Dynamixel Servos:** Connect the servos to the U2D2 controller. Ensure the correct wiring and that the servos' IDs match those in the code (e.g., `DXL1_ID` for pan and `DXL2_ID` for tilt).
- **Relay and GPIO:** For Linux systems, ensure the GPIO library (e.g., Jetson.GPIO) is correctly configured. The relay pin (e.g., `RELAY_PIN = 7`) should be wired to the relay hardware.

4.4 MQTT Broker Setup

- Install and run an MQTT broker such as Mosquitto.
- Update the MQTT topics and broker IP addresses in the code:
  * `MQTT_BROKER`
  * `MQTT_PORT`
  * All topic names (e.g., `dpad/commands`, `camera/feed`, etc.)

5 Usage Instructions

5.1 Running the Client Application

1. **Start the Application:** Run the client script from the command line:

```
python3 turret_client.py
```

2. **User Interface:** An OpenCV window titled "Turret Client" appears with:
   - The live camera feed.
   - A control panel on the left with an Auto/Manual toggle and speed slider.
   - A status message indicating the MQTT connection status.

3. **Controls:**
   - **Keyboard:** Arrow keys or `W`, `A`, `S`, `D` to adjust pan and tilt; Spacebar to trigger the relay (simulate firing); ESC to exit.
   - **Mouse:** Click on the control panel to toggle modes or adjust speed.

5.2 Running the Server Application

1. **Start the Application:** Run the server script from the command line:

```
python3 turret_server.py
```

2. **Hardware Initialization:** The server will attempt to initialize the camera, GPIO (for relay), and Dynamixel servos. Check the console logs for status messages.
3. **Operation Modes:**
   - **Manual Mode:** Direct commands from the client override servo positions.
   - **Auto Mode:** The system automatically adjusts the turret based on object detection (green objects).
4. **MQTT and Status Updates:** The server publishes camera frames and servo status updates to MQTT topics periodically.

5.3 User Interface and Controls

- **Control Panel:** Provides a visual interface for toggling auto/manual mode, adjusting the speed setting with a slider, and displaying MQTT connection status.
- **Visual Overlays:** The client displays crosshairs and bounding boxes (when a green object is detected) on the camera feed.

6 Code Structure and Key Functions

6.1 Client Code Overview

- **MQTT Callbacks:**
  * `on_connect(client, userdata, flags, reasonCode, properties)`: Handles successful MQTT connection and subscribes to necessary topics.
  * `on_message(client, userdata, msg)`: Processes incoming messages (camera frames, servo status, etc.).
- **Main Loop (`main_loop`):** Handles display updates, input events, and publishes control commands.
- **Input Handling:** Functions such as `mouse_callback` and key event handling in the main loop manage user interactions.
- **UI Rendering:** Functions like `draw_control_panel` and `draw_crosshair` create the visual elements in the OpenCV window.

6.2 Server Code Overview

– **Hardware Initialization:**

  * `initialize_camera()`: Sets up the camera feed.
  * `find_dynamixel_port()` and `initialize_servos()`: Detect and initialize Dynamixel servos.
  * `initialize_home_position()`: Establishes a reference "home" position for the servos.

– **Image Processing and Object Detection:**

  * `camera_feed_thread()`: Captures and processes video frames, performs object detection, and publishes the processed image.
  * `detect_green_objects(frame)`: Uses HSV thresholding to detect green objects and compute their bounding boxes.

– **Servo and Relay Control:**

  * `control_servos()`: Continuously updates servo positions based on either manual inputs or automatic corrections.
  * `write_servo_position(dxl_id, position)`: Sends position commands to the servos with error handling and retries.
  * `trigger_relay()`: Activates the relay with debounce logic.

6.3 Threading and Concurrency

Multiple threads are used for:

– **Camera Feed Processing:** Capturing, processing, and publishing video frames.

– **Servo Control Loop:** Continuously reading sensor data and updating servo positions.

– **MQTT Communication:** Running the MQTT client loop in a background thread.

Synchronization is managed using locks (e.g., `frame_lock`, `command_lock`) to ensure thread-safe operations.

7 Error Handling and Logging

7.1 Logging Configuration

– The code uses Python's built-in `logging` module for debug and informational messages as well as error reporting with stack traces.

– Logging levels (e.g., DEBUG, INFO, ERROR) are set to provide detailed feedback during operation.

7.2 Exception and Error Handling

– **Try/Except Blocks:** Most functions include try/except blocks to catch exceptions, log errors, and allow the system to continue operating.

– **Retries:** Functions such as `write_servo_position` and `read_servo_position` implement retry mechanisms to handle transient communication failures.

7.3 Troubleshooting Tips

– **MQTT Connection Issues:** Check that the broker IP and port are correctly configured and that the broker is running.

– **Camera Feed Problems:** Verify that the correct camera (or OAK-D device) is connected and accessible.

– **Servo Communication:** Ensure that the Dynamixel servos are properly wired to the U2D2 controller and that their IDs match those specified in the code.

– **GPIO and Relay:** For Linux systems, ensure that GPIO libraries (e.g., Jetson.GPIO) are correctly installed and that the relay wiring is correct.

8 Future Enhancements and Known Limitations

8.1 Future Enhancements

– Additional sensor integration (e.g., depth sensors, TOF sensors).

– Improved UI/UX for the control panel.

– Enhanced error recovery and logging mechanisms.

– Integration with additional input devices (e.g., game controllers).

8.2 Known Limitations

– Some hardware-specific features are only available on Linux.

– The camera initialization may require further adjustments based on device models.

– The current auto-mode sensitivity and correction parameters might need fine-tuning for different environments.

9 Appendices

9.1 MQTT Topics and Payload Formats

Topic	Direction	Payload Format	Description
`dpad/commands`	Client → Server	JSON (e.g., `{"pan_delta": 5}`)	Directional commands
`relay/command`	Client → Server	String (e.g., `"off"`)	Relay control
`camera/feed`	Server → Client	JPEG bytes	Camera feed
`camera/bbox`	Server → Client	JSON	Bounding boxes
`servo/status`	Server → Client	JSON	Servo positions
`server/tof`	Server → Client	JSON	ToF data
`server/sensitivity`	Both	JSON	Sensitivity

9.2 Library Versions and Dependencies

– **Python:** 3.7 or higher

– **OpenCV:** 4.x

– **NumPy:** 1.18 or higher

– **Paho MQTT:** 1.5.x

– **Pygame:** 2.x

– **Additional Libraries:** Refer to the installation instructions above for a complete list.

9.3 References and Resources

– Paho MQTT Documentation

– OpenCV Documentation

– Dynamixel SDK Documentation

– DepthAI Documentation

10 Contact and Support

For further assistance, bug reports, or feature requests, please contact:

– **Email:** `support@yourcompany.com`

– **Git Repository:** `https://github.com/MAVProxyUser/Gelblaster_Wingman`

– **Issue Tracker:** `https://github.com/MAVProxyUser/Gelblaster_Wingman/issues`

End of Documentation