ChatGPT

Gemini Robotics-ER Quickstart Adaptation for a Self-Control System

This document summarizes how to adapt examples from the Gemini Robotics-ER quickstart to the autonomous control system described in the specifications.md file. The goal is to make use of standard output formats, planning capabilities and progress estimation available in the Gemini robotics models to improve reliability and flexibility in your dashboard application.

1. Normalize outputs and adhere to JSON schema

The quickstart specifies that spatial outputs such as 2D points and bounding boxes should be represented as JSON objects with coordinates normalized between 0–1000. For example, a prompt might request that the model "Point to the bin on the shelf that needs restocking" and specify that the answer should be returned as a list of objects in the form `{\"point\": [y,x], \"label\": ...}` with coordinates normalized to 0–1000 [1]. Similarly, bounding boxes are returned in the format `[y_min, x_min, y_max, x_min]` with all coordinates normalized [2]. Adopting this convention in your Vision System allows you to reuse prompt examples and parsing code directly from the quickstart.

- Return bounding boxes and points as JSON arrays of objects with keys such as `"box_2d"` or `"point"`.
- Use the order `[y, x]` for points and `[y_min, x_min, y_max, x_max]` for boxes.
- Normalize each coordinate to the range 0–1000 regardless of image resolution; multiply by image dimensions when drawing overlays.
- When no objects are found, return an empty array instead of null.

2. Using ER as a secondary or primary planner

Gemini Robotics-ER is not limited to object detection; it can perform high-level reasoning, task planning and even call external tools. The quickstart demonstrates prompts where the model generates a sequence of points representing a trajectory, or returns a list of bounding boxes together with free-form explanations [3]. This suggests two integration strategies:

- **Planner-assist mode.** When your existing Flash-based planner cannot decide on the next step, send the current camera frame and recent action history to the ER model. The model returns a JSON with an action name and rationale such as `{"action": "move_forward_short", "reason": "target is far and centered"}`.
- **Tool-calling mode.** ER can call predefined functions through the tool-calling interface. Define the robot's actions as callable tools (e.g., `scan()`, `turn_left(angle)`, `move_forward(distance)`) and let the model propose sequences of calls to achieve a task.

3. Progress estimation and failure detection

One of ER's unique features is its ability to estimate the progress of a task. The blog explains that developers can use a "thinking budget" to trade off latency and reasoning quality [4]. To detect when the robot is stuck or the target is lost, periodically send recent frames and the action history to the ER model

and parse a JSON response such as `{"status":"IN_PROGRESS","progress":0.7,"comment":"target slightly right"}`. If progress remains below a threshold for several cycles, trigger a re-scan or re-plan automatically. This mechanism also allows you to display a progress bar and textual feedback in the UI.

4. Coordinate transformation between camera and robot frame

While the current implementation of `approach` uses only the bounding box height and horizontal center, you can leverage ER's normalized coordinates and MuJoCo's camera parameters to compute approximate world positions. Retrieve the intrinsic and extrinsic camera matrices from MuJoCo, convert the normalized `[y,x]` coordinates back to pixel positions, and perform an inverse projection to find where the target lies on the floor plane. Use this to compute a turning angle and forward distance rather than relying solely on servoing.

5. Prompt design patterns

The quickstart examples emphasise clear instructions: specify the output format, normalization range, and behaviour when no objects are found [5]. Use similar patterns when writing prompts for your Vision System and planner. For example:

- **Object detection prompt:** "Detect the red pole in the image. Return a JSON array of objects with keys `"box_2d"` and `"label"`. Coordinates should be normalized between 0 and 1000. If no poles are found, return an empty array."
- **Progress estimation prompt:** "Given these last 4 frames and the actions taken, estimate the progress toward reaching the red pole. Return a JSON object with keys `"status"` (IN_PROGRESS, STALLED, COMPLETED), `"progress"` (0–1), and `"comment"` explaining the status."
- **Planner prompt:** "Given the current robot view and the last action, choose the next action from [scan, turn_left_30, turn_right_30, move_forward_short, approach, stop]. Return a JSON object `{\"action\": <action_name>, \"reason\": <brief rationale>}`."

6. Pseudocode examples

The following pseudocode sketches illustrate how to integrate normalized bounding boxes, progress estimation and planner assistance into your system. They are deliberately abstract and should be adapted to your codebase.

```
# Normalize Vision System output
def process_bbox(bbox_norm, img_width, img_height):
    y_min_norm, x_min_norm, y_max_norm, x_max_norm = bbox_norm
    # Convert normalized coordinates to pixel positions
    y_min = y_min_norm / 1000 * img_height
    x_min = x_min_norm / 1000 * img_width
    y_max = y_max_norm / 1000 * img_height
    x_max = x_max_norm / 1000 * img_width
    return (y_min, x_min, y_max, x_max)
```

```
# ER planner-assist mode
def get_next_action(flash_planner, er_model, observation, history):
    action = flash_planner.plan(observation)
```

```
    if action is None:  # Flash is stuck
        er_input = {"image": observation, "history": history}
        er_output = er_model.call(er_input)
        return er_output["action"], er_output["reason"]
    return action, "flash planner output"
```

```
# Progress estimation and failure detection
def monitor_progress(er_model, frames, actions, threshold=0.2):
    er_input = {"frames": frames, "actions": actions}
    status = er_model.estimate_progress(er_input)
    progress = status.get('progress', 0)
    if progress < threshold:  # Considered stalled
        return "STALLED", status
    return status.get('status'), status
```

```
# Convert a normalized point to world coordinates using MuJoCo camera matrices
import numpy as np
def normalized_point_to_world(pt_norm, img_width, img_height, camera_intrinsics,
camera_extrinsics):
    y_norm, x_norm = pt_norm
    pixel_y = y_norm / 1000 * img_height
    pixel_x = x_norm / 1000 * img_width
    # Form a homogeneous image point
    img_point = np.array([pixel_x, pixel_y, 1.0])
    # Convert to camera coordinates via inverse intrinsics
    camera_point = np.linalg.inv(camera_intrinsics) @ img_point
    # Assume the target lies on the z=0 plane in world coordinates
    # Compute intersection of ray with plane using camera extrinsics (R,t)
    R = camera_extrinsics[:3, :3]
    t = camera_extrinsics[:3, 3]
    # Ray in world coordinates
    ray_dir = R.T @ camera_point
    camera_origin = -R.T @ t
    # Solve for scale s such that z_world=0: camera_origin_z + s*ray_dir_z = 0
    s = -camera_origin[2] / ray_dir[2]
    world_point = camera_origin + s * ray_dir
    return world_point[:2]  # x,y in world frame
```

References

1. Google Developers Blog, “Gemini 2.5 for robotics and embodied intelligence” – prompts specifying normalized point coordinates and JSON format [1].
2. Google Developers Blog, “Building the Next Generation of Physical Agents with Gemini Robotics-ER 1.5” – bounding box and trajectory prompts using normalized coordinates and JSON schemas [2] [3] [5].
3. Same source – discussion of thinking budgets and balancing latency with reasoning quality [4].

---

1 Gemini 2.5 for robotics and embodied intelligence - Google Developers Blog
https://developers.googleblog.com/en/gemini-25-for-robotics-and-embodied-intelligence/

2 3 4 5 Building the Next Generation of Physical Agents with Gemini Robotics-ER 1.5 - Google Developers Blog
https://developers.googleblog.com/en/building-the-next-generation-of-physical-agents-with-gemini-robotics-er-15/