-- Detailed Section ---

6.3 Detailed CSC and CSU Descriptions Section

This is the CURRENT software structure for the keyboard-controlled robotic arm system with external COCO object detection. The overall CSCI now comprises two primary Computer Software Components (CSCs) and supporting utilities:

- Perception CSC (COCO object detection)
- Arm Control CSC (keyboard teleoperation of the SO-101 arm)

Each CSC consists of one or more Computer Software Units (CSUs), which are Python modules with functions or classes.

6.3.1 Detailed Class Descriptions

6.3.1.1 VisionSystem

- **Purpose:** Manages a USB camera and performs object detection using a pretrained COCO SSD MobileNet v3 model.
- **Key Responsibilities:**
  - Initialize and configure the camera
  - Capture image frames
  - Perform object detection with OpenCV DNN
  - Optionally filter detections to a target label list
  - Draw detection overlays
- **Key Fields:**
  - classNames (labels from coco.names)
  - net (cv2.dnn_DetectionModel)
  - configPath, weightsPath
- **Key Methods**:
  - getObjects(img, thres, nms, draw=True, objects=[])
  - main loop: open camera, run getObjects(), display frames

6.3.1.2 DetectedObject

- **Purpose:** Represents an individual COCO detection returned by getObjects().
- **Key Fields:**
  - label, class_id
  - confidence
  - bbox
  - center_2d

6.3.1.3 ServoController (keyboard_motor_control.py)

- **Purpose:** Provides low-level communication with the Feetech STS3215 servos over the Lerobot MotorsBus.
- **Key Responsibilities:**

- Connect to /dev/ttyACM0 and instantiate MotorsBus with six joints
- Read present positions
- Send incremental Goal_Position commands per joint
- Basic canned “wave” starting pose sync_write

- **Key Fields:**
  - port (/dev/ttyACM0)
  - motors (six Motor definitions, MotorNormMode.RANGE_M100_100)
  - bus, connected, joint_names
- **Key Methods:**
  - motor_control(joint_number, offset)
  - pan_arm(), extend_shoulder(), extend_elbow(), flex_wrist(), twist_wrist(), hand_control()
  - wave_hand()
  - cleanup()

6.3.1.4 KeyboardTeleop (keyboard_motor_control.py main loop)

- **Purpose:** Maps keyboard keys to joint motions for manual teleoperation.
- **Key Responsibilities:**
  - Read single-key input from stdin
  - Graceful shutdown on ESC / Ctrl-C
- **Key Controls:**
  - a/d pan, w/s shoulder, y/h elbow, i/k wrist flex, j/l wrist twist, q/e gripper, n wave

6.3.1.5 PositionSequencer (simple_position_recorder.py) (TOOK REFERENCE)

- **Purpose:** Records and plays back joint position sequences over a Feetech motor bus.
- **Key Responsibilities:**
  - Connect to bus and remove limits
  - Torque on/off for manual posing
  - Record positions with timing
  - Replay recorded trajectories with timing
- **Key Methods:**
  - connect(), disconnect(), torque_on(), torque_off()
  - get_positions(), move_to_position()
  - record_sequence(name), play_sequence(name)

6.3.2 Detailed Interface Descriptions

Overview of how the current components interact:

- VisionSystem -> Any consumer
  - Transfers: 2D detections (label, bbox, confidence)
- KeyboardTeleop -> ServoController
  - Transfers: Keypress-triggered joint offsets; reads present positions and sends Goal_Position
- External assets -> VisionSystem

- Transfers: coco.names, ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt, frozen_inference_graph.pb

6.3.3 Detailed Data Structure Descriptions

Joint Command:

- Joint name or id plus an integer Goal_Position offset

Detection:

- label
- confidence
- bounding box
- 2D pixel center

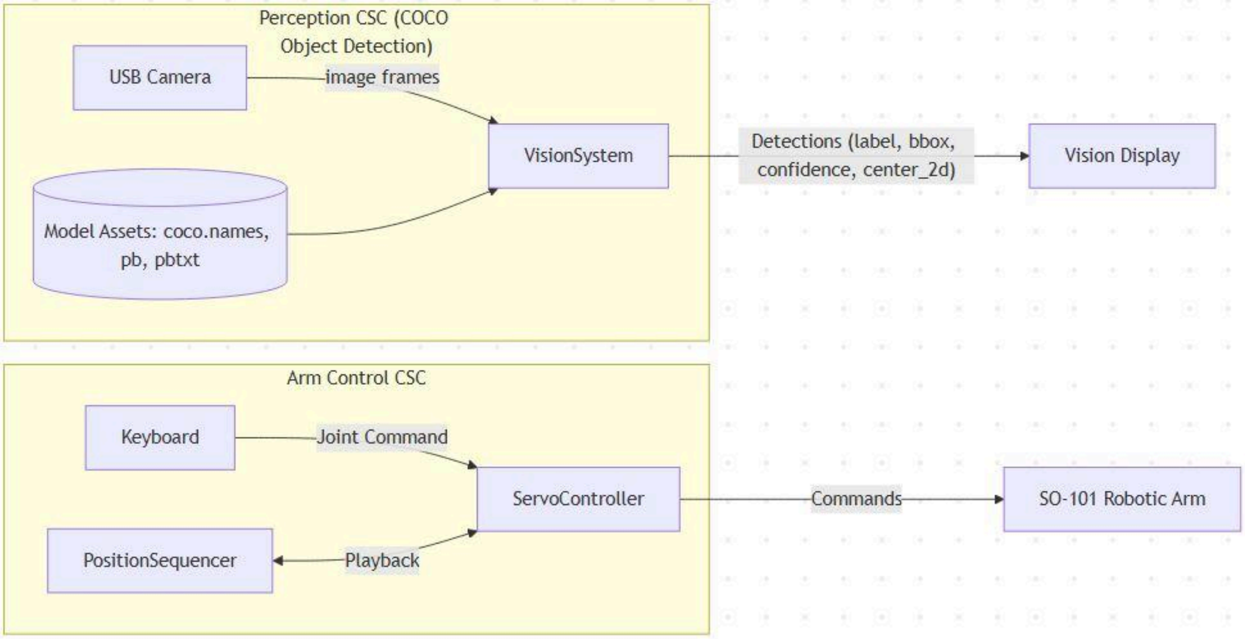
Sequence File (PositionSequencer):

- name, recorded_at, total_positions
- sequence: list of {position, positions{motor: value}, duration}

Model Assets:

- coco.names (labels)
- frozen_inference_graph.pb (COCO SSD MobileNet v3 weights)
- ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt (model config)

6.3.4 Detailed Design Diagrams



6.4 Database Design and Description Section

- No database for our project