---------- **Detailed Section** ----------

6.3 Detailed CSC and CSU Descriptions Section

This is the CURRENT software structure for the Vision-Guided Robotic Arm system. The overall CSCI is comprised of four major Computer Software Components (CSCs):

1. Perception & Depth Estimation CSC
2. Kinematics & Arm Control CSC
3. Calibration CSC
4. Coordination & Application CSC

Each CSC will consist of one or more Computer Software Units (CSUs), which are typically Python modules with one main class.

6.3.1 Detailed Class Descriptions

The following sections provide descriptions for the primary classes used in the Vision-Guided Robotic Arm system. These classes are grouped by CSU and listed with their key responsibilities, fields, and methods.

6.3.1.1 VisionSystem

- **Purpose:** Manages the USB camera and performs object detection using a COCO-trained model. Provides processed detections (i.e. detection results) to other key components.
- **Key Responsibilities:**
    - Initialize and configure the camera
    - Capture image frames
    - Perform object detection
    - Load camera calibration
    - Provide visualization of detection results
- **Key Fields:**
    - camera_id
    - camera_matrix, dist_coeffs
    - net
    - confidence_threshold
- **Key Methods:**
    - initalize_camera()
    - capture_frame()
    - detect_objects()
    - visualize_detections()
    - load_calibration()

6.3.1.2 DetectedObject

- **Purpose:** Represents an individual COCO detection
- **Key Fields:**
    - label, class_id
    - confidence
    - bbox

    - center_2d

6.3.1.3 DepthEstimator

- **Purpose:** Converts 2D detections into 3D coordinates using hybrid depth estimation methods.
- **Key Responsibilities**:
    - Estimate depth from object size, table geometry, or other assumptions
    - Project 2D pixel coordinates into 3D space
- **Key Fields:**
    - camera_matrix
    - object_database
    - table_height
- **Key Methods:**
    - estimate_depth_hybrid()
    - estimate_depth_from_size()
    - pixel_to_3d()

6.3.1.4 DepthEstimate

- **Purpose:** Holds depth estimation values (i.e. value, confidence)

6.3.1.5 ArmKinematics

- **Purpose:** Implements forward and inverse kinematics for the 6-DOF robotic arm
- **Key Responsibilities:**
    - Represents DH parameters
    - Compute end-effector pose from joint angles
    - Solve inverse kinematics numerically
    - Enforce joint limits
- **Key Methods:**
    - forward_kinematics()
    - inverse_kinematics()
    - check_collision()
    - get_workspace_bounds()

6.3.1.6 DHParameters

- **Purpose:** Stores Denavit-Hartenberg parameters for a single joint.

6.3.1.7 ServoController

- **Purpose:** Provides low-level communication with the STS3215 servo motors via the serial bus.
- **Key Responsibilities:**
    - Connect to and communicate over /dev/ttyACM0 (Raspberry Pi 4 port)
    - Send positional commands to servos
    - Enable or disable torque
    - Retrieve servo state
    - Execute multi-join movements
- **Key Methods:**

    - connect() && disconnect()
    - set_position()
    - set_positions()
    - get_servo_state()
    - emergency_stop()
    - home_position()

6.3.1.8 ServoState

- **Purpose:** Represents the current physical status of a servo (i.e. position, velocity, temperature, voltage, etc)

6.3.1.9 CameraCalibration

- **Purpose:** Performs intrinsic camera calibration using a checkerboard and produces a .npz file containing calibration parameters
- **Key Methods:**
    - capture_calibration_image()
    - calibrate()
    - save_calibration()

6.3.1.10 HandEyeCalibration

- **Purpose:** Computes the transform between the camera and the robot claw using ArUco marker detections and saved robot positions.
- **Key Purpose:**
    - Capture positions
    - Run OpenCV's calibrateHandEye() function
    - Save and load the transform matrix
- **Key Methods:**
    - capture_pose_pair()
    - calibrate()
    - transform_point_camera_to_robot()
    - save_calibration()
    - load_calibration()

6.3.1.11 VisionGuidedArm

- **Purpose:** Serves as the high-level controller that ties all of the other components together, to allow tasks such as object detection and pick and place.
- **Key Responsibilities:**
    - Initialize other components
    - Detect a target object
    - Estimate its 3D position
    - Run inverse kinematics to generate motion waypoints
    - Command the servo controller to execute to position
- **Key Methods:**
    - initialize()
    - detect_target()

    - estimate_target_3d_position
    - plan_grasp_motion()
    - execute_trajectory()
    - pick_and_place()
    - run_interactive_mode()

6.3.1.12 main.py

- **Purpose:** Provides the command line interface and program entry point.

6.3.2 Detailed Interface Descriptions

Overview for how we want the components to work with each other:

- VisionSystem -> DepthEstimator
    - Transfers: 2D detections (label, bounding box)
    - Output: Estimated depth and 3D coordinates
- Calibration -> Vision / Depth Estimation
    - Transfers camera intrinsics and distortion parameters
    - Depth estimation depends on these values
- Calibration -> Kinematics & Servo Control
    - Transfers claw <-> camera transform for coordinate alignment
- Coordinator -> Vision
    - Requests image frames and detection results
- Coordinator -> DepthEstimator
    - Converts detections into 3D coordinates for planning
- Coordinator -> Kinematics
    - Provides target 3D positions and receives joint angle solutions
- Coordinator -> ServoController
    - Sends joint positions to execute grasps and movements
- main.py -> Coordinator / Calibration
    - Passes user-selected operating mode and configuration settings

6.3.3 Detailed Data Structure Descriptions

Joint Configuration:

- A list of six floating-point joint angles, representing a full arm pose

3D Point

- A 3-element vector [x, y, z] in frame

Detections

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

Calibration Files

- File storing camera_matrix and dist_coeffs
- JSON file storing saved arm positions

6.3.4 Detailed Design Diagrams