KindaCodelessArm

By: Dylan Suzuki, Isaiah Pajarillo, Aidan Hodges, Jackson Muller

SDL Outline: https://bjohnson.lmu.build/cmsi4071web/overview.html

1. Project Proposal:

The project that our group hopes to complete for this class is a robotic arm that uses computer vision to perform some kind of task. As of right now, we don't have a concrete idea for what the task might be. Some possibilities include a sorter that removes recycling or compost from landfill-bound trash, an arm that recognizes facial structure and attempts to feed the user, or a desk organizer that recognizes common items and organizes them on your desk. We intend to use additive manufacturing to manage hardware costs, which would require some simple CAD knowledge and access to the EDC. Right now, we imagine using a rPi hooked up to some servo motors to drive the arm. The easiest way to implement the software would likely be using python to control the GPIO pins of the rPi based on some camera input, likely an inexpensive webcam. For the camera processing, we would probably have to learn OpenCV, TensorFlow, or some similar libraries for object/face recognition. Another key feature that we would need to both learn and implement is inverse kinematics. In order for our robotic arm to move to the locations we need it to, we would likely use numpy or some similar math library to implement any inverse kinematics.

We think this project is appropriate for this class due to its difficulty, opportunity to expand on topics we are interested in, and overall coolness. During our initial discussions on project topics, robotics came up quickly as a point of interest for us. Due to our focus on computer science, we hoped to find a project that would make use of interesting hardware and give us ample opportunity to write interesting software, without making the hardware unnecessarily complicated from a technical/EE perspective. We came up with a simple robotic arm due to its versatility and relative simplicity. However, we still think the hardware will be a considerable challenge since none of us have extensive robotics experience. On the software side, we see this project as a great way to extend our knowledge from classes like AI, Operating Systems, and Algos. In addition this project will allow us to gain valuable experience working together on a multidisciplinary project that integrates software, hardware and theoretical knowledge. We plan to approach each aspect of the build together, to ensure that everyone in the group gains exposure to the hardware design, control software, and computer vision components. Through tackling challenges side by side, we will strengthen our ability to problem-solve as a team, which helps mirror the collaborative environment we're expecting to

encounter in the industry. Another reason this project is a strong fit for the class is its flexibility. The robotic arm gives us the option to start with a straightforward application, such as detecting and picking up basic objects, and then gradually increasing complexity as we make progress. This flexibility helps ensure that our project scope can expand or contract depending on our timeline, while still allowing us to demonstrate creativity as well as technical ambition. In addition, it also reduces the risk of becoming blocked at a single obstacle, since alternate applications or simplified goals can always remain viable.

2. Project Proposal Document and Presentation Slides

Project Proposal Document
Proposal Presentation Slides

4. Software Development Plan

4.0 Software Development Plan

4.1 Plan Introduction

The project, *Vision Guided Robotic Arm*, is a 3D-printed, servo-driven robotic arm powered by a Raspberry Pi and webcam. It uses computer vision and inverse kinematics to detect, pick up, and manipulate objects. The rationale is to create an affordable, flexible, and educational robotic platform that integrates software and hardware in a single system

Development Activities:

- Hardware assembly and testing (3D printing, servo setup, Raspberry Pi integration)
- Software Development (Python + OpenCV + NumPy for object recognition and inverse kinematics)
- Integration and Testing (combining hardware and software)
- Iterative refinements

Milestones:

- Week 2: Project Proposal Presentations
- Week 5: Requirements + Preliminary Schedule Documents
- Week 8-9: Status Updates SCRUM
- Week 10: Software Design Description Document (Architecture Section)
- Week 12: Software Design Description Document (Detailed Section)
- Week 13-14: ABCDR Project Presentations
- Week 15: Final Presentations

4.1.1 Project Deliverables

- Week 2: Project Proposal Presentations
    - Short presentation introducing our project, goals, and audience
- Week 5: Requirements + Preliminary Schedule Documents
    - Organization of functional, performance, and environment requirements

        - Early milestones and rough schedules
    - Week 8-9: Status Updates SCRUM
        - Summarize progress on hardware assembly and initial vision software tests
    - Week 10: Software Design Description Document (Architecture Section)
        - Document detailing the high-level architecture of the system
        - Prototype design expected
    - Week 12: Software Design Description Document (Detailed Section)
        - Explanation of how kinematics will be implemented and how software modules will communicate
        - Hardware/Software integration should be underway
    - Week 13-14: ABCDR Project Presentations
        - Presentation of nearly finished arm
        - Demonstrates vision-guided pick-and-place action with object detection, trajectory planning, and servo actuation.
    - Week 15: Final Presentations
        - Full demo of robotic arm and capabilities

4.2 Project Resources

4.2.1 Hardware Resources

- **Execution:**
    - SO-ARM100 – GitHub - TheRobotStudio/SO-ARM100: Standard Open Arm 100
        - Servos – https://www.walmart.com/ip/seort/16716556565
        - Motor control board – https://www.amazon.com/Waveshare-Integrates-Control-Circuit-Supports/dp/B0CTMM4LWK/
        - Power supply – https://www.amazon.com/Facmogu-Switching-Transformer-Compatible-5-5x2-1mm/dp/B087LY41PV/
        - USB-C cables – https://www.amazon.com/Charging-etguuds-Charger-Braided-Compatible/dp/B0B8NWLLW2/?th=1
    - Webcam
    - Raspberry Pi
    - 3D printer parts
- **Development:**
    - 3D printer
    - Development computers (personal laptops)

4.2.2 Software Resources

- Source Code: https://github.com/brukg/SO-100-arm/tree/main/launch

- Python
- Opencv
- Numpy
- VSCode
- Github

4.3 Project Organization

Name	Email	Role
Aidan Hodges	ahodges4@lion.lmu.edu	Currently Undefined
Dylan Suzuki	dsuzuki@lion.lmu.edu	Currently Undefined
Isaiah Pajarillo	ipajaril@lion.lmu.edu	Currently Undefined
Jackson Muller	jmuller7@lion.lmu.edu	Currently Undefined

4.4 Project Schedule

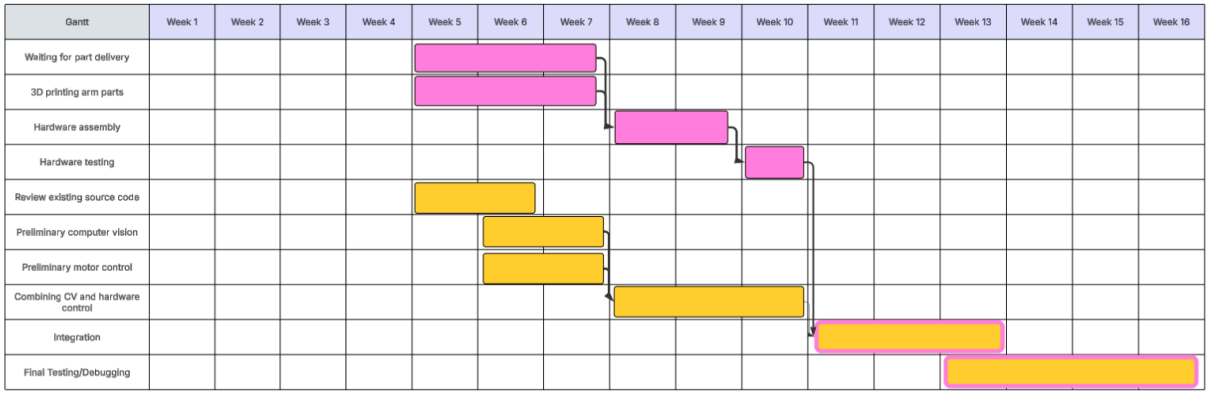
This section provides schedule information for the Robotic Arm= project

4.4.1 PERT / GANTT Chart

Hardware Tasks

Software Tasks

Robotic Arm Gantt Chart



https://lucid.app/lucidspark/936614d2-6ff0-4eaa-b5a8-3ba914b4f2fa/edit?viewport_loc=2414%2C826%2C4241%2C2404%2CuDe-dIt-NWfS&invitationId=inv_54860a9f-67dd-4b44-893b-698ab2df7bf5

4.4.2 Task / Resource Table

Task	Resource

Waiting for Part Delivery	n/a
3D Printing Arm Parts	3D printer, PLA filament
Hardware Assembly	Raspberry Pi, Servo motors, Motor driver, Power Supply
Hardware Testing	Raspberry Pi, Servos, Webcam
Review Existing Source Code	Laptops
Preliminary Computer Vision	Raspberry Pi, Webcam
Preliminary Motor Control	Raspberry Pi, Servo motors
Combining CV + Motor Control	Raspberry Pi, servos, webcam
Integration	Full robotic arm system
Final Testing/Debugging	Full robotic arm system

5. Requirements Specification (SRS) - Vision Guided Robotic Arm

5.1 Introduction

Description: We're creating a 3D-printed, servo-driven robotic arm powered by a Raspberry PI that uses a combination of a webcam and computer vision to detect and manipulate small objects. The system will ideally implement a pick-and-place movement action. Our arm is going to emphasize a CS-focused backend (Python + OpenCV + NumPy IK) with TensorFlow Lite. Users are going to be students, hobbyists, researchers, etc. Maintenance primarily consists of re-printing parts, replacing servos, and updating Python dependencies.

5.2 CSCI Component Breakdown

- 5.2.1 Vision - Webcam capture, processing, detection/classification
    - 5.2.1.1 Camera
    - 5.2.1.2 Detection Software

- 5.2.2 Motion & Control
    - 5.2.2.1 Trajectory Planner
    - 5.2.2.2 Servo Driver
    - 5.2.2.3 Safety Mechanisms (overheat protection, etc)

- 5.2.3 Logic
    - 5.2.3.1 Pick and Place
    - 5.2.3.2 Calibration (servos, ect)

- 5.2.4 User Interface
    - 5.2.4.1 CLI (start/stop, status, etc)
- 5.2.5 Data
    - Dataset (model training)

5.3 Functional Requirements

- 5.3.1 Vision
    - 5.3.1.1 The system shall acquire frames from a USB webcam at >= 15 FPS
    - 5.3.1.2 The system shall detect at least one simple object (i.e. ball, book, etc)
    - 5.3.1.3 The system shall provide the position of the detected object in the frame

- 5.3.2 Hardware
    - 5.3.2.1 The arm shall be able to move each joint when sent a basic command
    - 5.3.2.2 The system shall provide a way to stop all movement quickly (i.e. emergency stop)
    - 5.3.2.3 The system shall allow us to set a safe range of motion for each joint

- 5.3.3 Trajectory
    - 5.3.3.1 The system shall let us move the arm to a specific position
    - 5.3.3.2 The system shall move smoothly from one position to another without it being jumpy
    - 5.3.3.3 The system shall support a “rest” position command

- 5.3.4 Logic
    - 5.3.4.1 The system shall support a simple “pick and place” motion: detect -> move arm -> close hand -> move -> release
    - 5.3.4.2 The system shall allow switching between a manual mode (we control the arm with commands) and an automatic mode (self autonomy)

- 5.3.5 User Interaction
    - 5.3.5.1 The system shall provide a basic way to start and stop tasks from the terminal
    - 5.3.5.2 The system shall print simple status messages (e.g. “object found”, “no object found”, etc)

5.4 Performance Requirements

- 5.4.1 Response Time
    - 5.4.1.1 The system shall start moving the arm within a couple of seconds (1-4 seconds) after detecting an object.
- 5.4.2 Pick and Place Time
    - 5.4.2.1 The system shall be able to finish a simple pick and place motion in about 15 seconds or less.
- 5.4.3 Camera Performance

- 5.4.3.1 The system shall show camera images at a reasonable speed (10 fps or more) so that the arm can react

- 5.4.4 Accuracy
  - 5.4.4.1 The arm shall get close enough to the object to grab it (within about 1-4 cm of the target)
- 5.4.5 Safety
  - 5.4.5.1 When the stop command is executed, the arm shall stop moving right away

5.5 Project Environment Requirements

5.5.1 Development Environment

- 5.5.1.1 Operating System: Raspberry Pi OS 64-bit
- 5.5.1.2 Language: Python 3.12 (possibly more)
- 5.5.1.3 Libraries: OpenCV, NumPy
- 5.5.1.4 Tools: Github

5.5.2 Execution

- 5.5.2.1 Hardware
  - 5.5.2.1.1 Raspberry Pi 4
  - 5.5.2.1.2 USB Webcam (preferably 1080p)
  - 5.5.2.1.3 6x STS3215 Servos
  - 5.5.2.1.4 Motor Control Board
  - 5.5.2.1.4 Power Supply (5V)
  - 5.5.2.1.5 3D printed arm

6. Architectural Design Document

6.1 Introduction

This document presents the architecture and software design for the Vision-Guided Robotic Arm project. The system integrates computer visions and inverse-kinematics to allow our 3D printed robotic arm to detect, track, and interact with physical objects. The perception subsystem uses real time video footage from a USB webcam connected to Raspberry Pi 4, processed through NumPy and OpenCV for object detection. The control subsystem translates coordinates into motion commands while the actuation subsystem uses six servos to execute movements.

6.1.1 System Objectives

The objective of this system is to provide a vision-guided robotic arm that can detect, identify, and move physical objects with minimal user input. The system integrates computer vision and inverse kinematics to be able to recognize objects in the camera's field of view, and calculate the needed servo movements required to accurately grab and place objects. Objects of various shapes and colors are identified through an open computer vision library (OpenCV), and their positions are translated into coordinates for the robotic arm. Once the arm receives the coordinates, the arm will use inverse kinematics to accurately move its arm towards the desired object.

6.1.2 Hardware Interface

6.1.2.1 USB Webcam Interface

The Perception Subsystem uses a standard 2.0 camera connected directly to Raspberry Pi 4. It captures real-time 720p video frames at 30 fps.

6.1.2.2 Raspberry Pi 4 Interface

The Raspberry Pi 4 functions as the primary controller. It connects to the USB webcam for video capture and communicates with the motor-control board, which drives the servo motors.

6.1.2.3 Servo and Motor Control Board Interface

The motor-control board receives motion commands from the Raspberry Pi and transmits them to the STS3215 servos. Each servo is wired to a specific joint of the robotic arm allowing for physical movement.

6.1.2.4 Power Supply Interface

The system is powered by a DC power adapter that outputs 5V DC, which is suitable for the Raspberry PI and motor control components. The adapter connects through a barrel connector to the control board.

6.1.2.5 Development Hardware Interfaces

Development and debugging occur on personal laptops connected to the Raspberry Pi. These devices are used to upload Python scripts, run tests, and manage source code through GitHub.

6.1.2.6 3D-Printed Arm Assembly Interface

The 3D printed arm structure the physical framework for the robotic system. Its design was sourced from an open-source GitHub repository (SO-ARM100) to simplify assembly. The printed components house the servo motors and provide joints and linkages.

6.1.3 Software Interface

6.1.3.1 Programming Language Interface

All software is written in Python, running on Raspberry Pi.

6.1.3.2 Computer Vision Interface

The OpenCV library is used for object detection and image processing. Input frames from the webcam are passed through OpenCV functions.

6.1.3.3 Mathematical Computation Interface

NumPy is used to support matrix operations and inverse-kinematics calculations that determine servo positions.

6.1.3.4 Development Environment Interface

Development occurs within Visual Studio Code on personal laptops. The system's source code is stored in a GitHub Repo.

6.1.4 Human Interface

6.1.4.1 Developer Interface

Team members interact with the system through terminal commands on development computers or directly via Raspberry Pi 4.

6.1.4.2 Demonstration Interface

For demonstration purposes, users can observe the robotic arm's operation as it performs pick-and-place actions.

6.2 Architectural Design

The system architecture for the *Vision-Guided Robotic Arm* consists of three primary subsystems. These subsystems include:

1) Perception Subsystem
    - Captures real time video footage using a USB webcam connected to a Raspberry Pi 4.
    - Uses OpenCV and NumPy for object detection, color separation, and positional object analysis.
    - Outputs object coordinates in a normalized format to the control subsystem.
2) Control Subsystem
    - Implements inverse kinematics algorithms to calculate servo angles based on object coordinates.
    - In Python, this module interfaces with the motor control board to send position and motion commands.
    - Handles movement error correction, to ensure accurate motion trajectories.
3) Actuation Subsystem (Servo & Motor Hardware)
    - Includes six STS3215 servos, and a motor controller board.
    - Receives instructions and executes corresponding mechanical motion.
    - Provides feedback to control subsystem for calibration and adjustments.

6.2.1 Major Software Components

1) Vision Component:
    - Captures video frames, performs color filtering and object detection using OpenCV. Outputs the object coordinates.
2) Kinematics Component:
    - Takes object coordinates and converts it into servo angles using inverse kinematics calculations.
3) Control Component:
    - Translates calculated angles into PWM commands for the motor controller.
4) Calibration Component:
    - Aligns servo's to their zero positions
5) Main Coordinator Component:
    - Handles startup and shutdown sequences

6.2.2 Major Software Interactions

The vision-guided robotic arm will have 5 major software interactions:

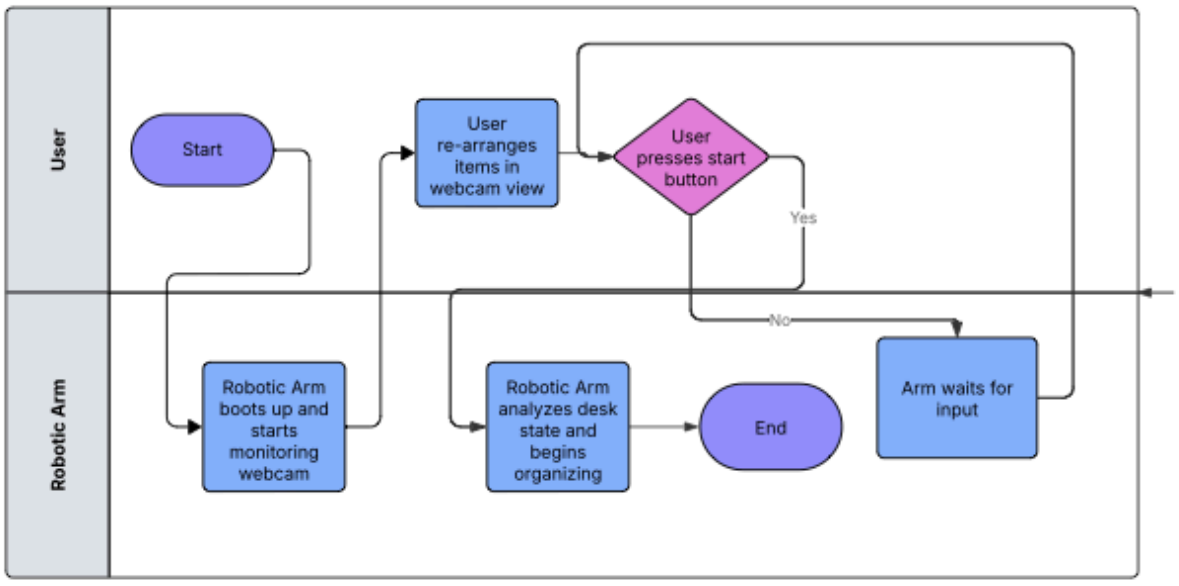
1. Vision to Kinematics:
    - The vision part outputs object coordinates as array: [x, y]. The kinematics part will take in these coordinates, and return a list of target servo angles.
2. Kinematics to Control:

- The control portion receives the computed angles as a list [θ_0, θ_1, θ_2, θ_3, θ_4, θ_5]. These are translated into PWM values and issues I^2C commands to the motor board.

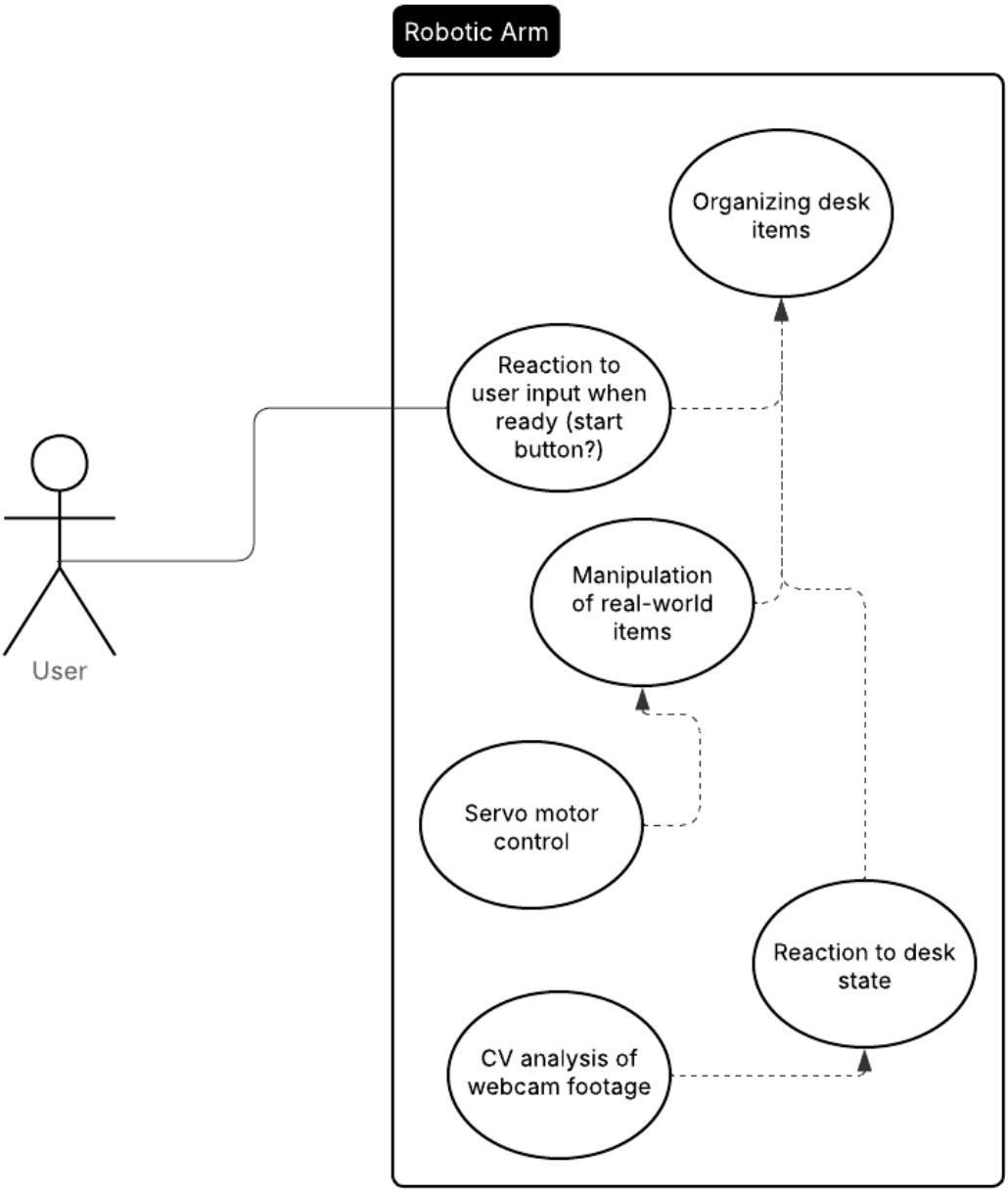
3. Control and Hardware:
    - Communication between the Raspberry Pi and the PWM board uses I^2C protocol. Ideal latency will be < 3 ms per command.
4. User Interaction:
    - The OpenCV display window runs on a separate thread to maintain responsive input handling and video feedback.

6.2.3 Architectural Design Diagrams

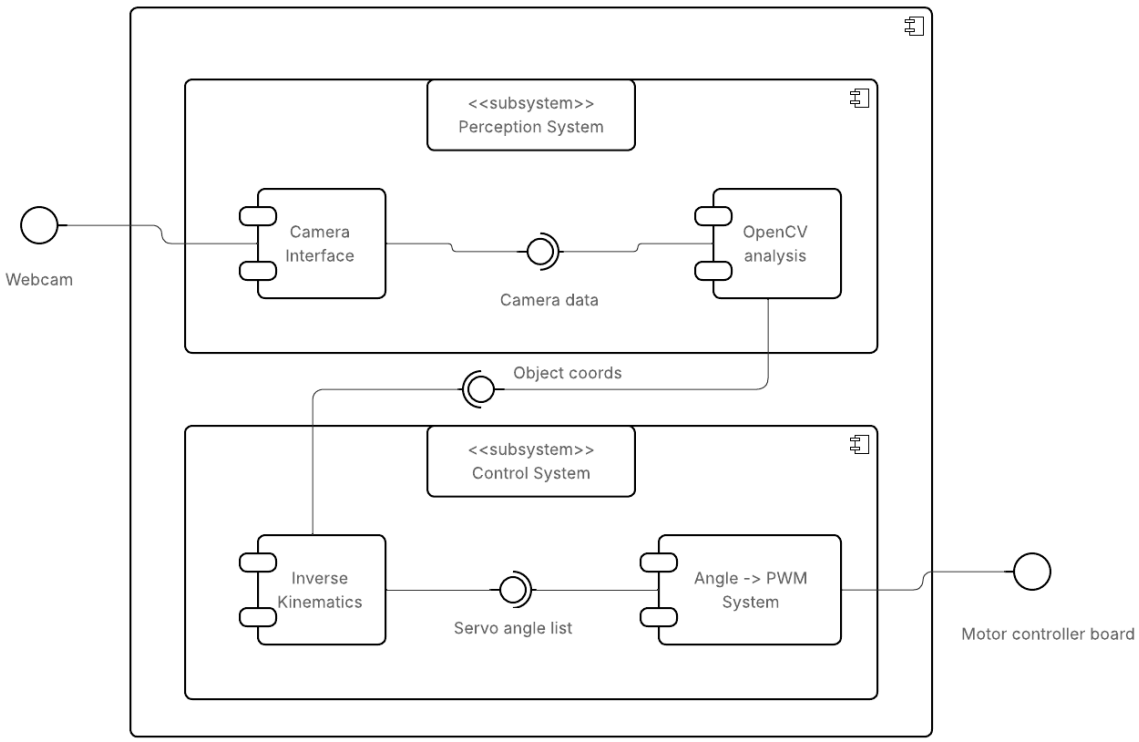
Swimlane Diagram



Use Case Diagram



Software Component Diagram



---------- 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

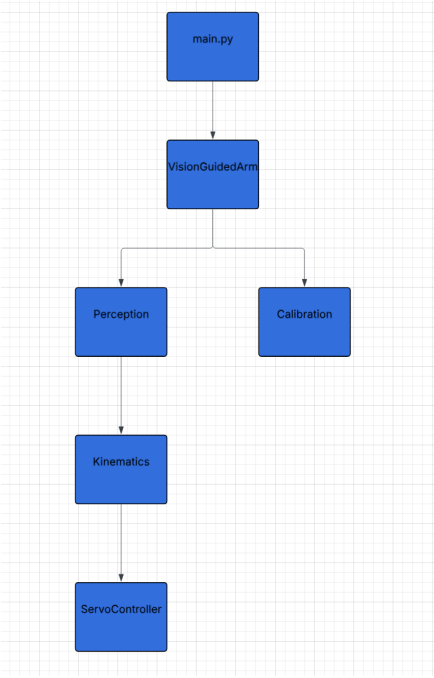
Detections

- 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



10. Testing

Due to the heavy emphasis on hardware this semester, we do not have a complete unit test plan, we have included video/images of us testing the arm in the Keck Lab in the repo.