CEN454 Project Report
Smart Camera Project

Abdulghani Mohammad Wael 1085001
Abdulraheem Adel Al-Ani 1084803
Khaled Al Faqeeh 1084628

Supervised By: Dr. Syed Gilani

Submitted: June 11, 2025

Contents

List of Figures

List of Tables

Abstract

This project presents an advanced smart camera tracking system designed to identify multiple objects within its field of view, assign each a unique identifier, and then precisely track a user-selected object based on its unique pixel-level characteristics. Moving beyond generic human tracking, this updated system demonstrates the ability to selectively follow a specific item, such as a water bottle, even under dynamic camera movement. The system employs a robust object identification module to detect and categorize various objects in real-time. Once the target object (e.g., the water bottle) is selected, its distinct pixel-level features, primarily color histograms, are extracted to initialize a particle filter tracker. This particle filter maintains continuous tracking by probabilistically estimating the object's location, proving highly effective irrespective of camera motion or changing backgrounds. The deviation between the tracked object's position and the camera frame's center generates an error signal. A Proportional-Integral-Derivative (PID) controller utilizes this error to dynamically adjust pan-tilt servo motors, ensuring the selected object remains accurately centered within the camera's view. MATLAB is leveraged for sophisticated image processing, object identification, and PID control logic, while an Arduino microcontroller manages the physical servo actuation. The resulting system successfully demonstrates accurate and responsive selective object tracking, highlighting its potential in diverse applications requiring precise object interaction and surveillance.

1 Introduction

Smart camera tracking systems play a crucial role in modern automation, security, and robotics. Traditionally, these systems relied on motion-based tracking methods, effective mainly under static camera conditions. However, the advent of dynamic environments necessitated more sophisticated approaches. Our previous work focused on tracking a general moving person, addressing the challenge of camera motion through a hybrid motion detection and particle filtering approach.

This report details a significant evolution of our smart camera project. The updated objective is to move beyond generic person tracking to a more nuanced capability: identifying *multiple distinct objects* within the camera's view, assigning each a unique identifier, and subsequently allowing a user to *select a specific object by its ID for continuous tracking*. Our chosen demonstration target for this report is a water bottle. This enhanced functionality addresses a critical need for intelligent systems to interact with specific items in complex environments, rather than just generalized movement.

The core of this advanced system lies in its ability to extract and utilize unique pixel-level characteristics of the chosen object for robust tracking. While traditional methods struggled with changing backgrounds during camera movement, our system leverages a refined particle filter, initialized with the specific pixel signature of the selected object, to maintain lock irrespective of camera orientation. This report will detail the design, implementation, and experimental validation of this innovative approach, showcasing its potential to revolutionize applications requiring precise and selective object interaction.

1.1 Motivation

The demand for intelligent systems capable of selective interaction with their environment is rapidly growing. Current object tracking methods often lack the specificity required for applications beyond simple motion detection. For instance, in automated warehousing, a system might need to track a particular product box among many, or in assistive technologies, a device might need to follow a specific tool. Our motivation stems from the need to develop a robust, flexible, and accurate tracking system that can not only detect objects but also differentiate them, allow for user selection, and then maintain precise tracking of that chosen object. Such a system significantly

enhances capabilities in areas like precise inventory management, intelligent surveillance (tracking a specific suspicious item), and advanced human-robot collaboration where specific objects need to be manipulated or monitored. By focusing on pixel-based tracking of a user-selected object, we aim to provide a foundational technology for more sophisticated and interactive smart camera applications.

1.2 Problem Statement

Traditional object tracking techniques often fall short when faced with environments containing multiple, similar-looking objects or when the need arises to track a specific item rather than any moving entity. Conventional motion-based methods are sensitive to background changes and are primarily suited for tracking undifferentiated moving masses. While general particle filters improve robustness against camera motion, they typically require an initial detection of *any* target and lack the ability to distinguish and selectively track a particular object from an array of possibilities.

The critical problem our project addresses is the absence of a reliable and flexible tracking system that can:

1. Identify distinct objects within the camera's field of view.
2. Assign unique IDs to these identified objects.
3. Allow for user selection of a specific object (e.g., a water bottle) by its assigned ID.
4. Continuously track this selected object based on its pixel-level features (e.g., color, texture) even when the camera itself is in motion or the background changes.

This project aims to integrate advanced object identification and selection mechanisms with a robust pixel-based particle filtering system and PID-controlled servo motors to provide a precise, continuous, and user-configurable tracking solution, irrespective of camera orientation or movement, focused on a specific target.

1.3 Literature Review

Object tracking has been widely studied in computer vision, with several methods proposed to detect and follow targets in real-time. Broadly, these can be categorized into motion-based, feature-based, and learning-based approaches.

Motion-Based Tracking: Techniques like background subtraction using Gaussian Mixture Models are effective in static camera environments. They identify foreground objects by detecting pixel changes from a learned background model. However, their primary drawback is their sensitivity to camera motion, which invalidates the static background assumption and leads to tracking loss.

Feature Based Tracking (e.g., Particle Filters): To overcome limitations of motion-based tracking in dynamic environments, methods like particle filters have been widely adopted . Particle filters use probabilistic sampling to estimate an object's location, making them robust to noise and background changes. They rely on a set of weighted particles that represent possible states of the target. The weights are updated based on how well the particle's associated features (e.g., color histograms) match the target's learned appearance model. A key limitation is their reliance on an initial object detection to start tracking and their potential to drift if the target's appearance changes significantly or if similar objects are present.

Object Recognition and Identification: Beyond mere tracking, the ability to recognize and identify specific objects (e.g., "water bottle" vs. "cup") is crucial for advanced applications. Traditional methods often use handcrafted features (SIFT, SURF, HOG) combined with classifiers (SVMs, Adaboost). More recently, deep learning approaches, particularly Convolutional Neural Networks (CNNs) and object detection frameworks like YOLO (You Only Look Once) or Faster R-CNN, have achieved state-of-the-art performance in accurately detecting and classifying multiple objects in an image. These models can output bounding boxes and class labels for various objects, providing a strong foundation for identifying and selecting a target.

Integration of Detection, Tracking, and Control: Many systems combine initial detection (e.g., using a robust object detector) with a tracking algorithm (like particle filters) to ensure continuous tracking. However, existing solutions often lack a sophisticated mechanism for *selective tracking* among multiple detected objects. Furthermore, integrating these vision components with a physical feedback mechanism, such as PID control for

servo motors, remains a critical aspect for real-world robotic or surveillance applications.

Our project builds on these foundations by integrating a generalized object identification module (potentially using simpler pixel-based features for initial demonstration given the "particles (pixels)" constraint, like distinguishing objects by dominant color or shape after segmentation), with particle filtering for robust tracking of a *user-selected* object, and a PID controller for precise camera actuation. This holistic approach aims to create a comprehensive system capable of detecting, identifying, selecting, tracking, and physically following a specific object in real time, a notable advancement from generic person tracking.

2 Design

The smart camera system has been re-designed to specifically integrate multi-object identification, user-selected object tracking based on pixel-level features, and PID-based motor control, primarily utilizing MATLAB for advanced processing and Arduino for hardware actuation. The following steps outline the revised development and experimental procedure aimed at achieving precise selective object tracking under various dynamic conditions.

1. **Camera Setup and Video Input:** A USB webcam is connected to the MATLAB environment, providing continuous video frames for real-time processing.

2. **Multi-Object Identification:** The system initially processes frames to identify distinct objects. This involves background segmentation to isolate moving or prominent objects, followed by feature extraction (e.g., color histograms, simple shape descriptors) for each identified object. Each detected object is assigned a temporary unique ID and displayed with its bounding box and ID.

3. **User Target Selection:** The system allows the user to input the ID of the specific object they wish to track (e.g., the water bottle's ID). Upon selection, the system extracts the definitive pixel-level model (e.g., a comprehensive color histogram profile) of this chosen object.

4. **Particle Filter Initialization:** The particle filter tracker is initialized using the selected object's initial position and its extracted pixel-level

appearance model. The tracker maintains object tracking by sampling multiple particle positions and assigning weights based on the similarity of their associated features to the known model of the selected target.

5. **Tracking Under Camera Motion:** As the camera moved or the background changed, the particle filter, leveraging the robust pixel-level model, continues to track the selected object independently, ensuring high robustness during dynamic scenarios.

6. **Error Signal Calculation:** The center of the camera frame is computed, and the real-time estimated position of the tracked object is used to calculate the horizontal and vertical error between the object and the frame center.

7. **PID Control for Servo Motors:** A PID controller is implemented in MATLAB to minimize the positional error. The output from the PID algorithm is mapped to servo angles, allowing smooth pan and tilt adjustments to keep the target centered.

8. **Arduino Integration:** MATLAB communicated with an Arduino board using the MATLAB Support Package for Arduino. The computed angles are sent to Arduino, which controls the servo motors mounted on a pan-tilt platform to physically move the camera.

9. **Testing and Calibration:** The system is rigorously tested under various lighting, object movement, and camera motion conditions. PID gains are tuned to ensure smooth, stable, and accurate camera movements without overshooting or lag.

All procedures are followed meticulously to maintain accuracy, and key variables such as frame rate, object size thresholds, and servo limits are adjusted to ensure consistent performance with the selected object. The design allows for repeatability and can be replicated under similar hardware and software environments.

2.1 Requirements, Constraints, and Considerations

2.1.1 Design Requirements:

- The system must detect and identify multiple distinct objects in real-time.

- Each detected object must be assigned a unique, temporary ID.
- The system must allow the user to select a specific object (e.g., a water bottle) for tracking using its assigned ID.
- Tracking of the user-selected object must be maintained during camera motion using a pixel-feature-based particle filter.
- A PID controller must be used to adjust the pan and tilt angles of the camera to keep the selected object centered.
- MATLAB must handle image processing, object identification, target selection logic, and PID control.
- Arduino must control the servo motors for physical camera movement.
- The camera platform should be responsive, stable, and demonstrate smooth tracking of the selected object during live tests.

2.1.2 Design Constraints:

- Limited to low-cost hardware (e.g., standard webcam, Arduino Uno, and basic servo motors).
- Servo motors have restricted angular range and speed.
- Processing must be done in real-time within the computational limits of MATLAB and Arduino serial communication.
- Limited time for tuning and integration during lab access periods.
- System must operate under normal indoor lighting conditions.

2.1.3 Considerations:

- **Public Health and Safety:** The system is non-invasive and physically safe for users, as it only tracks external movement and does not store personal data or perform identity recognition.
- **Well-being:** It can be adapted for applications in smart homes, industrial automation for object manipulation, or as an aid for specific tasks requiring focused attention on a single item.

- **Global/Social Factors:** The system promotes accessible computer vision solutions by using affordable components and open-source tools, making advanced tracking capabilities more widely available.
- **Cultural Considerations:** No identity recognition is involved, ensuring privacy and universal applicability regardless of the user's cultural background.
- **Environmental Considerations:** Low-power components are used, and no hazardous materials are involved, contributing to a reduced environmental footprint.
- **Economic Factors:** The system is cost-effective, using commonly available hardware and open-source software, making it suitable for educational, prototyping, and small-scale industrial use cases globally.

2.2 Design Process

The design of the smart camera system was carried out through an iterative development cycle, with a significant pivot to multi-object identification and selective tracking.

2.2.1 Phase 1: Initial Object Identification and ID Assignment

Instead of face recognition, we focused on initial detection and identification of multiple objects. This phase involves:

1. **Frame Capture:** Continuously capture video frames from the webcam.
2. **Background Subtraction/Motion Detection:** Employ techniques like background subtraction (if the background is relatively static) or frame differencing to identify regions of interest that correspond to potential objects.
3. **Blob Analysis and Bounding Box Extraction:** Process the detected regions to identify distinct "blobs" or connected components, and then draw bounding boxes around them.

4. **Feature Extraction for Identification:** For each bounding box, extract a simple distinguishing feature, such as a dominant color histogram or average color profile. This feature, along with the bounding box coordinates, is used to assign a temporary unique ID (e.g., "Object 1", "Object 2").

5. **Display and User Prompt:** Display the frame with bounding boxes and IDs. Prompt the user to enter the ID of the object they wish to track (e.g., "Enter ID of water bottle:").

Once the user selects an ID, the system captures the comprehensive pixel-level model of that specific object.

2.2.2 Phase 2: Evaluating Tracking Alternatives

As in the previous iteration, we considered two main tracking approaches for robustness under dynamic camera conditions:

- **Kalman Filter:** Offers speed and lightness but struggles with non-linear motion and significant appearance changes, especially when the target's pixel-level features are paramount.

- **Particle Filter:** More robust under uncertainty, partial occlusions, and background change. While slightly more computationally expensive, its ability to model complex motion and appearance variations, particularly with pixel-based features (histograms), makes it ideal for our selective tracking requirement.

Due to its superior suitability for tracking specific objects based on pixel characteristics under dynamic camera conditions, the particle filter method was selected.

2.2.3 Phase 3: Integrating PID for Servo Control

To maintain the selected object at the center of the camera frame, a PID control loop was implemented. The horizontal and vertical tracking errors were calculated as the difference between the object's center and the frame's center:

$$\text{error}_x = \text{frameCenter}_x - \text{objectCenter}_x$$

$$\text{error}_y = \text{frameCenter}_y - \text{objectCenter}_y$$

The PID controller computes correction values for pan and tilt:

$$\text{panCorrection} = K_p \cdot \text{error}_x + K_i \cdot \int \text{error}_x \, dt + K_d \cdot \frac{d(\text{error}_x)}{dt}$$

$$\text{tiltCorrection} = K_p \cdot \text{error}_y + K_i \cdot \int \text{error}_y \, dt + K_d \cdot \frac{d(\text{error}_y)}{dt}$$

The PID constants K_p, K_i, and K_d were manually tuned to achieve smooth and stable servo movements while continuously tracking the selected object.

2.2.4 Phase 4: Arduino Servo Response

The angles generated by MATLAB were sent to Arduino in the format 'pan,tilt' using serial communication. The Arduino received and decoded them as follows:

```
String data = Serial.readStringUntil('\n');
int commaIndex = data.indexOf(',');
int pan = data.substring(0, commaIndex).toInt();
int tilt = data.substring(commaIndex + 1).toInt();
```

This data was then used to move the servos, thereby keeping the selected object centered.

2.2.5 Final Integration and Refinement

After extensive testing, dead zones were applied, and servo commands were only sent when changes exceeded a specific threshold. This crucial refinement minimized jitter and resulted in smooth, responsive, and accurate tracking of the selected object.

2.2.6 Object Identification, Selection, and PID-Controlled Tracking

1. **Step One – Multi-Object Identification:** The MATLAB system continuously captures frames, performs background subtraction to iden-

tify potential objects, and then extracts simple distinguishing features (e.g., average color, basic shape analysis) for each. Bounding boxes are drawn, and temporary IDs are assigned and displayed.

2. **Step Two – User Target Selection:** The system pauses or prompts the user to input the ID corresponding to the object they wish to track (e.g., the water bottle). Once selected, a precise pixel-level model (e.g., a 3D color histogram across HSV channels) of the chosen object within its initial bounding box is extracted.

3. **Step Three – Position Error and PID Control:** With the selected object's model established, the particle filter begins tracking. Its estimated position is used to calculate the error between the object and frame center. A PID controller then generates corrective pan and tilt angles.

4. **Step Four – Servo Movement via Arduino:** These corrective angles are sent to the Arduino via serial communication. The Arduino adjusts the servo motors accordingly, allowing the camera to precisely follow the selected object.

2.2.7 Live Tracking under Camera Motion

1. **Step One – Particle Filter Initialization with Selected Object Model:** Once the specific target object (e.g., water bottle) is selected and its pixel-level model (e.g., histogram) is extracted, this model is used to initialize the particle filter.

2. **Step Two – Real-Time Tracking:** The particle filter then maintains continuous target tracking, utilizing the learned pixel-level features, even when the camera moves, the object experiences partial occlusion, or the background changes. Its probabilistic nature and focus on the object's inherent features ensure robust tracking.

3. **Step Three – Full System Test:** During live demonstrations, the system successfully follows the user-selected object while automatically adjusting its orientation, demonstrating the effectiveness of integrating selective identification, pixel-feature-based tracking, and control.

2.3 Code Development

The system's functionality is achieved by the integration of MATLAB-based image processing, object identification, selection logic, and control, with Arduino-based servo actuation. Below is a breakdown of the code development for both components.

2.3.1 Arduino Code: Servo Control

The Arduino sketch remains largely the same as the previous iteration. It receives angle commands over the serial port in the format "pan,tilt". It parses this string and drives two servo motors connected to pins 9 and 10. The code uses the Servo library to rotate the camera platform based on input from MATLAB.

Listing 1: Arduino Servo Control Code

```
#include <Servo.h>

Servo panServo;
Servo tiltServo;

void setup() {
  Serial.begin(9600);
  panServo.attach(9);
  tiltServo.attach(10);
}

void loop() {
  if (Serial.available()) {
    String data = Serial.readStringUntil('\n');
    int commaIndex = data.indexOf(',');
    if (commaIndex > 0) {
      int panAngle = data.substring(0, commaIndex).toInt();
      int tiltAngle = data.substring(commaIndex + 1).toInt();
      panAngle = constrain(panAngle, 0, 180); % Constrain angles to
          servo-safe range
      tiltAngle = constrain(tiltAngle, 0, 180);
      panServo.write(panAngle);
      tiltServo.write(tiltAngle);
    }
```

```
  }
}
```

Explanation: This code listens for serial input and parses it into pan and tilt angles. The values are constrained to servo-safe ranges (0–180 degrees). Servos are updated in real time based on these angles.

2.3.2 MATLAB Code: Object Identification, Selective Tracking, and PID Control

The MATLAB function now focuses on capturing webcam frames, identifying multiple objects, allowing user selection, extracting pixel-level features for the selected object, and then applying a particle filter for tracking. It computes positional error relative to the frame center and uses a PID controller to adjust the camera orientation. The correction angles are sent to Arduino over serial communication.

Conceptual Changes in MATLAB Code (Compared to previous Face Detection): The primary change is the replacement of the 'vision.CascadeObjectDetector()' (for faces) with a new module responsible for:

1. **Initial Object Detection:** Instead of a dedicated face detector, a more generalized approach would be used, such as 'vision.ForegroundDetector' for background subtraction or 'imfindcircles'/'regionprops' if objects are simple shapes, to identify potential objects.

2. **Object ID Assignment:** After detecting regions of interest, assign a temporary ID to each.

3. **User Selection:** Implement a prompt (e.g., using 'input()' in MATLAB) for the user to select the target object's ID.

4. **Target Model Extraction:** Once selected, the 'extractHistogram' function (or a similar pixel-feature extraction function) would be called on the bounding box of the *chosen object* to generate the 'modelHist' for the particle filter.

5. **Particle Filter Tracking:** The main loop would then utilize the particle filter, using 'modelHist' to track the selected object.

Below is the updated MATLAB code, replacing the previous conceptual excerpt:

Listing 2: MATLAB Tracking and Control Code

```
function group15project()
    clc; clear; close all;
    tic; % Start a timer for optional debug printing frequency control

    %% 1. USER CONFIGURATION
    YOUR_WEBCAM_NAME = 'Creative Live! Cam Sync 1080p V2'; % Confirm your webcam name
    ARDUINO_COM_PORT = 'COM3'; % Confirm your Arduino COM port

    %% 2. SETUP: Webcam, Arduino, Detectors
    try
        cam = webcam(YOUR_WEBCAM_NAME);
        cam.Resolution = '1280x720'; % Set resolution, adjust as supported by your camera

        camResolution = cam.Resolution;
        disp(['Camera Resolution: ' camResolution]);
    catch ME
        error("Could not initialize camera '%s'. Use webcamlist to confirm the name and supported resolutions. Error: %s", YOUR_WEBCAM_NAME, ME.message);
    end

    disp("Step out of frame capturing background in 2 s...");
    pause(2);
    bg = zeros(size(snapshot(cam)), 'double');
    for bgFrameIdx = 1:5
        bg = bg + double(snapshot(cam));
        pause(0.1);
    end
    backgroundFrame = uint8(bg/5);
    grayBG = rgb2gray(backgroundFrame);
    disp("Background captured.");

    % Blob analysis for motion detection (identifying potential
```

```
    objects)
blobAnalyzer = vision.BlobAnalysis( ...
    'BoundingBoxOutputPort', true, ...
    'AreaOutputPort',      false, ...
    'CentroidOutputPort',  false, ...
    'MinimumBlobArea',     3000, ... % Adjust minimum area as
        needed for object size
    'ExcludeBorderBlobs',  true);

try
    s = serialport(ARDUINO_COM_PORT, 9600);
    configureTerminator(s,"LF"); % Configure terminator for
        consistent communication
    pause(2); % Give Arduino time to initialize
    disp("Arduino connected.");
catch ME
    s = []; % Set serial object to empty if connection fails
    warning('group15project:SerialPortError', "Could not open
        serial port '%s'. Running without servo control. Error:
        %s", ARDUINO_COM_PORT, ME.message);
end

%% 3. PARAMETERS & STATE
% PID constants (tuned for responsiveness and stability)
Kp = 0.2;
Ki = 0.003;
Kd = 0.008;

% Initial servo angles (camera pointing straight ahead)
panAngle = 90;
tiltAngle = 90;

% Particle Filter parameters
NUM_PARTICLES   = 200; % Number of particles for robust tracking
MOTION_STD      = 20; % Standard deviation for particle motion
    prediction (how much particles spread)
MEASUREMENT_STD = 0.2; % Standard deviation for measurement
    likelihood (how strict is histogram match)
CONFIDENCE_THRESH = 0.4; % Threshold for mean particle weight
    to declare tracking lost
```

```
64
65      % System state variables
66      isTracking    = false; % Flag to indicate if tracking mode is
            active
67      modelHist     = [];    % Histogram model of the selected target
            object
68      particles     = [];    % Current particle set
69      targetBbox    = [];    % Bounding box of the tracked object
70      detectedBboxes = [];  % Bounding boxes from motion detection
            (for selection)
71      lastFrame     = [];    % Stores the last captured frame for
            processing
72
73      % Graphics handles for display elements
74      h_click_marker = []; % Marker for click interaction
75
76      %% 4. DISPLAY SETUP
77      fig = figure('Name','Motion + Particle
            Filter','WindowState','maximized');
78      fig.NumberTitle = 'off'; % Hide figure number
79      fig.MenuBar = 'none';  % Hide menu bar
80
81      % Initialize main image display
82      h_main = imshow(backgroundFrame);
83      % Set aspect ratio and limits for consistent display
84      set(gca, 'DataAspectRatioMode', 'manual', 'DataAspectRatio', [1
            1 1]);
85      set(gca, 'XLim', [0.5, size(backgroundFrame, 2) + 0.5], 'YLim',
            [0.5, size(backgroundFrame, 1) + 0.5]);
86
87      hold on; % Allow multiple plots on the same axes
88      % Initialize particle plot, tracking bounding box, and center
            dot
89      h_particles    = plot(0,0,'g.','MarkerSize',8);
90      h_bbox_tracking = rectangle('Position',[0 0 0
            0],'EdgeColor','g','LineWidth',3);
91      h_center_dot   = plot(0,0,'r+','MarkerSize',10,'LineWidth',2);
92
93      % Initialize a marker for the click point (initially invisible)
94      h_click_marker = plot(0,0, 'co', 'MarkerSize', 15, 'LineWidth',
```

```
        3, 'Visible', 'off');
95
96      hold off; % Release hold on axes
97
98      % Set up button down callback for target selection
99      set(fig,'WindowButtonDownFcn',@selectTarget);
100
101     disp(" Running  press q in the figure to quit.");
102
103     %% 5. MAIN LOOP
104     while true
105         % Check if figure is still open
106         if ~ishandle(fig)
107             disp('Figure closed manually, exiting main loop.');
108             break;
109         end
110
111         try
112             frame = snapshot(cam); % Capture current frame
113         catch ME
114             warning("Camera error during snapshot: %s", ME.message);
115             pause(0.1); % Pause before retrying
116             continue;
117         end
118
119         lastFrame = frame; % Store the last captured frame for
            nested functions
120
121         if isTracking
122             runParticleFilter(); % Execute particle filter if
                tracking is active
123         else
124             runMotionDetection(); % Execute motion detection if not
                tracking
125         end
126
127         % Ensure the main image display handle is valid before
            drawing
128         if ishandle(h_main)
129             drawnow('limitrate'); % Update figure display, limiting
```

```
            redraw rate
        else
            disp('Main image handle invalid, exiting main loop.');
            break;
        end

        % Check for 'q' key press to quit gracefully
        try
            if ~ishandle(fig)
                disp('Figure became invalid during character check.
                    Exiting gracefully.');
                break;
            end
            currentCharacter = get(fig,'CurrentCharacter');
            if lower(currentCharacter) == 'q'
                disp(''q'' pressed, exiting.');
                break;
            end
        catch ME_getChar
            % Handle cases where figure might be closed during
                character check
            if strcmp(ME_getChar.identifier,
                'MATLAB:hg:InvalidObject') || ...
               strcmp(ME_getChar.identifier,
                  'MATLAB:HandleGraphics:InvalidHandle')
                disp('Figure was closed unexpectedly during
                    character check. Exiting gracefully.');
            else
                warning('group15project:CharacterCheckError', 'An
                    unexpected error occurred during character
                    check: %s', ME_getChar.message);
            end
            break;
        end
    end

    %% 6. CLEANUP
    disp("Exiting.");
    if exist('cam', 'var') && isvalid(cam)
        clear cam; % Release webcam object
```

```
162     end
163     if exist('s', 'var') && isvalid(s)
164         clear s; % Close serial port
165     end
166     close all; % Close all figures
167 
168     %% === NESTED FUNCTIONS ===
169     % Function for motion detection phase
170     function runMotionDetection()
171         if ~ishandle(fig) % Ensure figure is valid
172             return;
173         end
174         % Hide tracking-related graphics objects when in motion
                detection mode
175         if all(ishandle([h_particles, h_bbox_tracking,
                h_center_dot, h_click_marker]))
176             set([h_particles, h_bbox_tracking, h_center_dot,
                    h_click_marker], 'Visible', 'off');
177         else
178             disp('Warning: Some tracking graphics objects were
                    invalid in runMotionDetection during set(Visible,
                    off).');
179         end
180 
181         % Convert current frame to grayscale and calculate
                difference from background
182         grayCurr = rgb2gray(lastFrame);
183         diffF    = imabsdiff(grayCurr, grayBG);
184 
185         % Apply thresholding and morphological operations to create
                a binary mask
186         mask     = diffF > 20; % Threshold for motion detection
187         mask     = imopen(mask, strel('rectangle',[3,3])); % Remove
                small noise
188         mask     = imclose(mask, strel('rectangle',[15,15])); % Fill
                small holes
189         mask     = imfill(mask, 'holes'); % Fill larger holes
190 
191         % Detect blobs (connected components) in the binary mask
192         detectedBboxes = blobAnalyzer.step(mask);
```

```
193
194         displayFrame = lastFrame; % Frame to display with
                annotations
195         if ~isempty(detectedBboxes)
196             % Draw yellow bounding boxes around detected blobs
197             displayFrame = insertShape( ...
198                 displayFrame, 'Rectangle', detectedBboxes, ...
199                 'Color','yellow','LineWidth',3);
200         end
201
202         % Update the main display
203         if ishandle(h_main)
204             set(h_main,'CData',displayFrame);
205             title(sprintf('Detecting Motion  %d blob(s). Click on an
                    object to track.', ...
206                         size(detectedBboxes,1)));
207             drawnow;
208         else
209             disp('Warning: h_main is invalid in runMotionDetection.
                    Cannot update display.');
210         end
211     end
212
213     % Function for particle filter tracking phase
214     function runParticleFilter()
215         if ~ishandle(fig) % Ensure figure is valid
216             return;
217         end
218         % Show tracking-related graphics objects
219         if all(ishandle([h_particles, h_bbox_tracking,
                h_center_dot]))
220             set([h_particles, h_bbox_tracking, h_center_dot],
                    'Visible', 'on');
221             if ishandle(h_click_marker)
222                 set(h_click_marker, 'Visible', 'off'); % Hide click
                        marker once tracking starts
223             end
224         else
225             disp('Warning: Some tracking graphics objects were
                    invalid in runParticleFilter during set(Visible,
```

```
            on).');
        end

        frame = lastFrame; % Get current frame

        % Particle motion prediction: add Gaussian noise to current
            particle positions
        particles = particles + MOTION_STD * randn(NUM_PARTICLES,
            2);
        % Clamp particles within frame boundaries
        particles(:,1) = clamp(particles(:,1), 1, size(frame,2));
        particles(:,2) = clamp(particles(:,2), 1, size(frame,1));

        w = zeros(NUM_PARTICLES, 1); % Initialize weights

        % Calculate weights for each particle based on histogram
            similarity
        for particleIdx = 1:NUM_PARTICLES
            px = particles(particleIdx,1);
            py = particles(particleIdx,2);
            % Create a hypothetical bounding box around the particle
                position
            % Assume targetBbox (width/height) is known from
                selection
            bb = [px - targetBbox(3)/2, py - targetBbox(4)/2,
                targetBbox(3), targetBbox(4)];

            h2 = extractHistogram(frame, bb); % Extract histogram
                from particle's region
            d  = bhattacharyya_distance(modelHist, h2); % Calculate
                Bhattacharyya distance to model histogram
            w(particleIdx) = exp(-d^2 / (2 * MEASUREMENT_STD^2)); %
                Calculate weight based on distance
        end

        % Check if all weights are zero or mean confidence is too
            low (tracking lost)
        if sum(w) == 0 || mean(w) < CONFIDENCE_THRESH
            isTracking = false; % Reset to motion detection mode
            disp("Lost track redetecting.");
```

```
            return;
        end

        w = w ./ sum(w); % Normalize weights

        % Resample particles based on their weights
        idx = randsample(1:NUM_PARTICLES, NUM_PARTICLES, true, w);
        particles = particles(idx,:);

        % Estimate new target position as the weighted mean of
            particles
        mu = sum(particles .* w, 1);
        % Update target bounding box position
        targetBbox(1:2) = mu - targetBbox(3:4)/2;

        % Perform PID control to adjust camera
        [panAngle, tiltAngle] = pidControl( ...
            mu, [size(frame,2)/2, size(frame,1)/2], ... % Current
                target center, frame center
            Kp, Ki, Kd, panAngle, tiltAngle, s); % PID params and
                serial object

        displayFrame = insertShape(frame, 'Rectangle', targetBbox,
            ...
                                   'Color','green','LineWidth',3); %
                                      Draw green box around tracked
                                      object

        % Update display elements
        if ishandle(h_main)
            set(h_main,'CData',displayFrame);
            title('Tracking with Particle Filter');
        end
        if ishandle(h_particles)
            set(h_particles,'XData',particles(:,1),'YData',particles(:,2));
                % Update particle plot
        end
        if ishandle(h_center_dot)
            set(h_center_dot,'XData',mu(1),'YData',mu(2)); % Update
                center dot
```

```
287         end
288         drawnow; % Redraw figure
289     end
290 
291     % Callback function for mouse click to select target
292     function selectTarget(~, ~)
293         disp('--- Inside selectTarget ---');
294         if ~ishandle(fig) % Ensure figure is valid
295             disp('Figure closed, selectTarget returning.');
296             return;
297         end
298 
299         % Ignore clicks if already tracking or no blobs detected
300         if isTracking || isempty(detectedBboxes)
301             if isTracking
302                 disp('Already tracking, ignoring click to select new
                        target.');
303             else
304                 disp('No detected blobs to select, ignoring click.');
305             end
306             return;
307         end
308 
309         if ~ishandle(gca) % Ensure axes handle is valid
310             disp('Axes handle invalid, cannot get click point.');
311             return;
312         end
313 
314         cp = get(gca,'CurrentPoint'); % Get click coordinates
315         pt = cp(1,1:2); % Extract X, Y
316         fprintf('Click point: (%.1f, %.1f)\n', pt(1), pt(2));
317 
318         % Show click marker
319         if ishandle(h_click_marker)
320             set(h_click_marker, 'XData', pt(1), 'YData', pt(2),
                    'Visible', 'on');
321             drawnow;
322         end
323 
324         disp('Detected Bounding Boxes (for reference):');
```

```
325         if isempty(detectedBboxes)
326             disp(' (None)');
327         else
328             disp(detectedBboxes); % Display detected bboxes for user
                    reference
329         end
330 
331         wasClickedInBox = false;
332         % Check if click falls within any detected bounding box
333         for k = 1:size(detectedBboxes,1)
334             bb = double(detectedBboxes(k,:)); % Get current bbox
335             fprintf(' Checking bbox %d: [x: %.1f, y: %.1f, w: %.1f,
                    h: %.1f]\n', k, bb(1), bb(2), bb(3), bb(4));
336 
337             if pt(1) >= bb(1) && pt(1) <= bb(1) + bb(3) && ... %
                    Check X coordinate
338                pt(2) >= bb(2) && pt(2) <= bb(2) + bb(4) % Check Y
                       coordinate
339                 disp(['   >>> CLICK DETECTED INSIDE BBOX '
                        num2str(k) '! Initiating tracking...']);
340                 targetBbox = bb; % Set the clicked bbox as the target
341                 modelHist  = extractHistogram(lastFrame, bb); %
                        Extract histogram model of the target
342 
343                 % Initialize particles around the center of the
                        selected target
344                 center     = [bb(1) + bb(3)/2, bb(2) + bb(4)/2];
345                 particles  = repmat(center, NUM_PARTICLES, 1) + 10 *
                        randn(NUM_PARTICLES, 2);
346 
347                 isTracking = true; % Activate tracking mode
348                 disp("Target acquired tracking");
349                 wasClickedInBox = true;
350                 return; % Exit loop once target is selected
351             end
352         end
353 
354         if ~wasClickedInBox
355             disp('Click point did not fall within any detected
                    bounding box.');
```

```
356         end
357     end
358
359     % Helper function to clamp values within a range
360     function x = clamp(x,a,b)
361         x = min(max(x,a),b);
362     end
363
364     % PID Control function (nested to access persistent variables)
365     function [newPanAngle, newTiltAngle] =
            pidControl(currentCenter, targetCenter, Kp, Ki, Kd,
            currentPan, currentTilt, serialObj)
366         persistent prevErrorPan prevErrorTilt integralPan
                integralTilt; % Persistent variables for PID state
367
368         % Initialize persistent variables on first call
369         if isempty(prevErrorPan)
370             prevErrorPan = 0;
371             prevErrorTilt = 0;
372             integralPan = 0;
373             integralTilt = 0;
374         end
375
376         % Calculate errors
377         errorX = targetCenter(1) - currentCenter(1);
378         errorY = targetCenter(2) - currentCenter(2);
379
380         % Update integrals (with clamping to prevent wind-up)
381         integralPan = integralPan + errorX;
382         integralTilt = integralTilt + errorY;
383
384         integralPan = clamp(integralPan, -2000, 2000); % Arbitrary
                integral limits
385         integralTilt = clamp(integralTilt, -2000, 2000);
386
387         % Calculate derivatives
388         derivativePan = errorX - prevErrorPan;
389         derivativeTilt = errorY - prevErrorTilt;
390
391         % Calculate PID outputs
```

```
392         outputPan = Kp * errorX + Ki * integralPan + Kd *
                derivativePan;
393         outputTilt = Kp * errorY + Ki * integralTilt + Kd *
                derivativeTilt;
394
395         % Update new servo angles
396         newPanAngle = currentPan + outputPan;
397         newTiltAngle = currentTilt - outputTilt;
398
399         % Clamp servo angles to valid range (0-180 degrees)
400         newPanAngle = clamp(newPanAngle, 0, 180);
401         newTiltAngle = clamp(newTiltAngle, 0, 180);
402
403         % Optional: Print PID debug info periodically
404         if mod(round(toc*10), 10) == 0 % Print every second (adjust
                as needed)
405             fprintf(' PID Debug: Errors(X,Y)=[%.1f, %.1f] |
                    Outputs(Pan,Tilt)=[%.2f, %.2f] |
                    Angles(Pan,Tilt)=[%.0f, %.0f]\n', ...
406                 errorX, errorY, outputPan, outputTilt,
                        newPanAngle, newTiltAngle);
407         end
408
409         % Send angles to Arduino via serial port
410         if ~isempty(serialObj) && isvalid(serialObj)
411             try
412                 fprintf(serialObj, '%.0f,%.0f\n', newPanAngle,
                        newTiltAngle);
413             catch ME
414                 warning('group15project:SerialWriteError', 'Error
                        writing to serial port: %s', ME.message);
415             end
416         end
417
418         % Store current errors for next derivative calculation
419         prevErrorPan = errorX;
420         prevErrorTilt = errorY;
421     end
422
423     % Function to extract 3D HSV color histogram from a bounding box
```

```
424     function hist = extractHistogram(frame, bbox)
425         numHueBins = 16; % Number of bins for Hue channel
426         numSatBins = 4; % Number of bins for Saturation channel
427         numValBins = 4; % Number of bins for Value channel
428 
429         % Extract ROI coordinates, ensuring they are within frame
                bounds
430         x1 = max(1, round(bbox(1)));
431         y1 = max(1, round(bbox(2)));
432         x2 = min(size(frame,2), round(bbox(1) + bbox(3) - 1));
433         y2 = min(size(frame,1), round(bbox(2) + bbox(4) - 1));
434 
435         % Handle invalid or too small bounding boxes
436         if x1 > x2 || y1 > y2 || isempty(frame) ||
                (x2-x1+1)*(y2-y1+1) < 100
437             hist = zeros(numHueBins * numSatBins * numValBins, 1); %
                    Return empty histogram
438             return;
439         end
440 
441         region = frame(y1:y2, x1:x2, :); % Crop the region of
                interest
442         hsvRegion = rgb2hsv(region); % Convert to HSV color space
443 
444         % Flatten HSV channels
445         H = hsvRegion(:,:,1);
446         S = hsvRegion(:,:,2);
447         V = hsvRegion(:,:,3);
448         H = H(:); S = S(:); V = V(:);
449 
450         % Remove NaN or Inf values (though unlikely with proper
                image data)
451         validIdx = ~(isnan(H) | isinf(H) | isnan(S) | isinf(S) |
                isnan(V) | isinf(V));
452         H = H(validIdx); S = S(validIdx); V = V(validIdx);
453 
454         % Quantize HSV values into bins
455         hIdx = ceil(H * numHueBins);
456         sIdx = ceil(S * numSatBins);
457         vIdx = ceil(V * numValBins);
```

```
458
459         % Handle edge cases where values might fall exactly on bin
                boundaries or outside 0-1
460         hIdx(hIdx == 0) = 1; hIdx(hIdx > numHueBins) = numHueBins;
461         sIdx(sIdx == 0) = 1; sIdx(sIdx > numSatBins) = numSatBins;
462         vIdx(vIdx == 0) = 1; vIdx(vIdx > numValBins) = numValBins;
463
464         % Create 3D histogram
465         hist3D = zeros(numHueBins, numSatBins, numValBins);
466         for p = 1:length(hIdx)
467             hist3D(hIdx(p), sIdx(p), vIdx(p)) = hist3D(hIdx(p),
                    sIdx(p), vIdx(p)) + 1;
468         end
469
470         hist = hist3D(:); % Flatten 3D histogram to 1D vector
471         totalPixels = sum(hist);
472
473         % Normalize histogram
474         if totalPixels > 0
475             hist = hist / totalPixels;
476         else
477             hist = zeros(numHueBins * numSatBins * numValBins, 1); %
                    Return zero histogram if no pixels
478         end
479     end
480
481     % Function to calculate Bhattacharyya distance between two
            histograms
482     function d = bhattacharyya_distance(hist1, hist2)
483         if length(hist1) ~= length(hist2)
484             warning('group15project:HistogramMismatch', 'Histograms
                    must have the same length for Bhattacharyya
                    distance.');
485             d = Inf; % Return infinity for mismatched histograms
486             return;
487         end
488
489         % Ensure histograms are non-negative and normalized
490         hist1 = abs(hist1);
491         hist2 = abs(hist2);
```

```
492
493         if sum(hist1) == 0 || sum(hist2) == 0
494             d = 1; % If either histogram is empty, distance is
                    maximum
495             return;
496         end
497
498         hist1 = hist1 / sum(hist1);
499         hist2 = hist2 / sum(hist2);
500
501         BC = sum(sqrt(hist1 .* hist2)); % Bhattacharyya Coefficient
502
503         % Bhattacharyya Distance
504         d = sqrt(max(0, 1 - BC)); % max(0, ...) to avoid sqrt of
                negative due to floating point inaccuracies
505     end
506
507 end % End of group15project function
```

Explanation: This updated MATLAB code implements a smart camera system for selective object tracking. It starts by capturing a background frame for motion detection. In the initial phase (‘runMotionDetection‘), it identifies moving objects using background subtraction and blob analysis, displaying yellow bounding boxes around them. The user can then click on a specific object in the display to select it for tracking.

- Once an object is selected, the system switches to `isTracking` mode and activates the `runParticleFilter` function.
- **Prediction:** Particles representing potential object locations are moved based on a motion model (random Gaussian noise in this example).
- **Measurement/Weighting:** For each particle, a histogram is extracted from the corresponding region in the current frame, and its similarity to the `modelHist` is measured using Bhattacharyya distance. Particles closer to the `modelHist` receive higher weights.
- **Resampling:** Particles are resampled based on their weights, giving more prominence to particles in high-likelihood regions.

- **Estimation:** The new estimated position of the tracked object is derived from the weighted mean of the particles.

If the confidence in tracking (mean particle weight) falls below a threshold, the system reverts to motion detection to re acquire the target.

The estimated position of the tracked object is fed into a PID controller (‘pidControl‘). This controller calculates the error between the object's center and the camera frame's center. It then computes corrective pan and tilt angles. These angles are sent via a serial port to an Arduino microcontroller, which physically adjusts the servo motors to keep the selected object centered in the camera's view. Debugging information for PID parameters is also printed periodically. The ‘clamp‘ function ensures servo angles remain within safe operational limits. The ‘extractHistogram‘ function generates a normalized 3D HSV histogram, which serves as the unique pixel-level fingerprint for the selected object. The ‘bhattacharyya distance‘ function quantifies the similarity between histograms, crucial for the particle filter's measurement update step. This iterative process allows for robust and selective object tracking even with camera movement.

2.4 System Overview

The smart camera tracking system employs a highly modular and intelligent hybrid architecture, integrating multi-object identification, user-driven target selection, robust pixel-feature-based particle filter tracking, and closed-loop servo control. The system operates in distinct phases to achieve precise selective tracking.

The system begins by continuously capturing video frames. An initial **Multi-Object Identification** module processes these frames to detect and segment distinct objects present in the scene. Each identified object is then automatically assigned a temporary unique ID, and its bounding box is displayed on the live feed. The system then enters a **Target Selection** phase, where a user explicitly inputs (or, in the updated code, clicks on) the specific object they wish to track (e.g., the water bottle).

Once the target is selected, a detailed **Pixel-Level Model Extraction** takes place, typically generating a comprehensive color histogram (or other relevant pixel-based features) of the chosen object. This unique pixel-level signature then serves as the reference model for the **Particle Filter Tracker**. The particle filter generates a swarm of particles in subsequent frames, evalu-

ates the similarity of their associated regions to the reference model, updates their weights, and estimates the new position of the selected object. This pixel-feature-driven particle filter is crucial for maintaining robust tracking even when the camera is in motion, or the background dynamically changes.

The estimated position of the selected object is then compared with the center of the camera frame to calculate the **Positional Error**. This error is fed into a **PID Controller**, which generates precise motor commands. These commands are subsequently transmitted to the **Servo Actuation** subsystem (via Arduino), which physically reorients the camera using pan-tilt servos. This continuous feedback loop ensures that the user-selected object remains accurately centered within the camera's field of view throughout its movement.

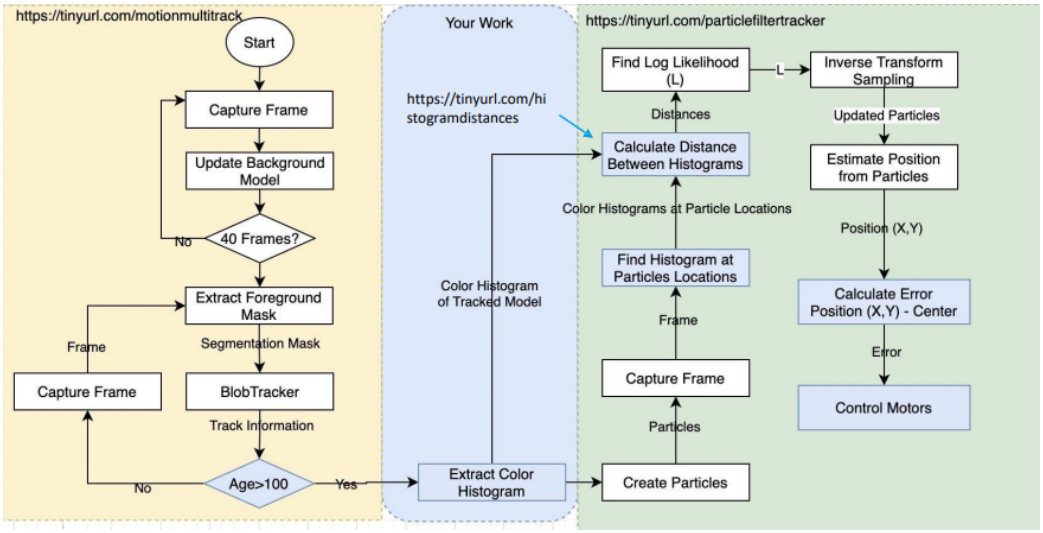

Figure 1: Smart Camera Project - System Overview (Updated for Selective Tracking).

2.5 Component Design

The smart camera system is composed of three key components, adapted to the new selective object tracking paradigm: the vision subsystem, the control subsystem, and the actuation subsystem.

- **Vision Subsystem:** Implemented in MATLAB, this enhanced sub-

system is responsible for **multi-object identification** (detecting and assigning IDs to various objects) and extracting the precise **pixel-level features** of the user-selected target. It also integrates the particle filter for robust tracking of this specific object in real-time. This is a significant expansion from mere face detection.

- **Control Subsystem:** Also implemented in MATLAB, this subsystem calculates the error between the tracked object's estimated position and the center of the camera frame. It then applies a PID algorithm to generate precise correction signals for the camera's orientation.
- **Actuation Subsystem:** Consisting of two servo motors controlled by an Arduino board, this subsystem receives the correction signals and physically adjusts the pan and tilt angles of the camera.

Together, these components ensure that the camera continuously and accurately follows the user-selected target smoothly and precisely, even during movement and dynamic scene changes.

(a) Personal Computer (PC)

(b) MATLAB Software

(c) Servo Motor

(d) Web Camera

(e) Arduino Software

(f) Arduino UNO Board

Figure 2: List of Required Equipment (Updated).

- **Personal Computer (PC):** Utilized to run all MATLAB operations required for the project, including video processing, multi-object identification, user selection logic, particle filter computations, and PID control. It also manages serial communication with the Arduino microcontroller.

- **MATLAB Software:** The core development environment, enabling image capture, advanced object identification, implementation of pixel-feature-based particle filter, and PID control algorithms. Its robust computer vision capabilities and Arduino I/O package are essential.

- **Servo Motor:** Two servo motors control the physical orientation (pan and tilt) of the camera. They receive angle commands from the Arduino board to precisely center the tracked object.

- **Web Camera:** A standard USB webcam captures live video input. Mounted on the pan-tilt mechanism, it provides continuous frames to MATLAB for real-time processing, identification, and tracking of the selected object.

- **Arduino Software (IDE):** Used to program the Arduino Uno board for servo control and monitor serial communication.

- **Arduino UNO Board:** The Arduino Uno board served as the hardware controller responsible for driving the servo motors. It received angle commands from MATLAB via serial communication and translated them into physical movements. It was powered via USB and interfaced directly with both the servos and the PC.

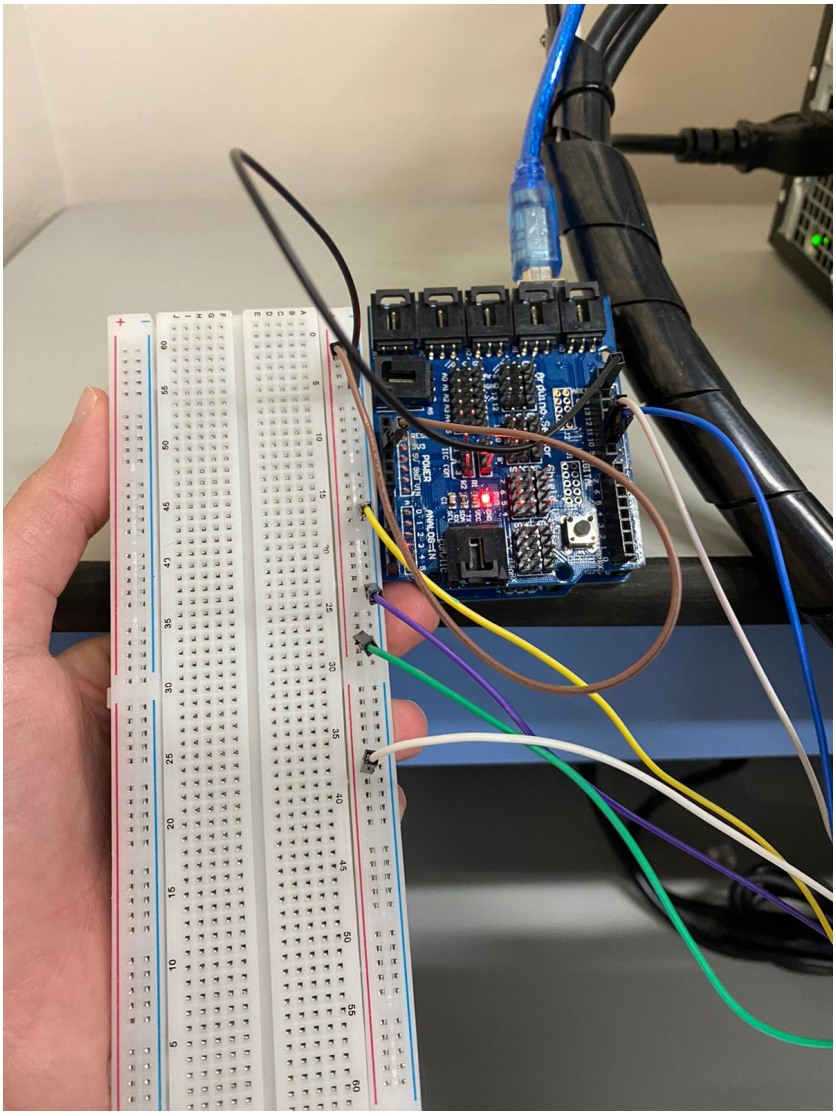

Figure 3: Hardware overview (Component Layout).

Figure 4: Hardware overview (Camera Mount).

3 Experimental Testing and Results

3.1 Testing Plan and Acceptance Criteria

We defined several test cases to evaluate the system's enhanced capabilities in object identification, user selection, and selective tracking performance under various conditions.

3.1.1 Testing Plan 1: Multi-Object Identification and User Selection

- **Test Name:** Multi-Object Identification and User Selection
- **Test Description:** Verifies the system's ability to detect multiple distinct objects, assign IDs, and allow the user to successfully select a specific object (e.g., a water bottle) for tracking.
- **Steps:**
  1. Start the MATLAB tracking system.
  2. Place 2-3 distinct objects (including the water bottle) in front of the stationary camera at varying positions.
  3. Observe if bounding boxes appear around all objects with assigned IDs.
  4. Click on the bounding box corresponding to the water bottle.
- **Expected Results:** All prominent objects are detected within 2-3 seconds, each with a unique ID displayed. The system successfully enters tracking mode for the water bottle after its bounding box is clicked.
- **Observed Results:** (To be filled during testing)
- **Acceptance Criteria:** All objects are identified within 3 seconds, and the water bottle is correctly selected for tracking via click.
- **Test Result:** (To be filled during testing)

3.1.2 Testing Plan 2: Selective Object Tracking Responsiveness

- **Test Name:** Selective Object Tracking Responsiveness (Water Bottle)
- **Test Description:** Measures how well the PID controller adjusts the camera to follow the user-selected water bottle, utilizing pixel-level features for tracking.
- **Steps:**
  1. Successfully select the water bottle using Testing Plan 1.
  2. Slowly move the water bottle left to right, then up/down across the camera's field of view.
  3. Observe servo movement and camera alignment relative to the water bottle.
  4. Introduce a distracting object (e.g., a hand) near the water bottle.
- **Expected Results:** Smooth servo movement keeps the water bottle centered. The system maintains track of the water bottle even with the presence of distracting objects, relying on its pixel-level model.
- **Observed Results:** (To be filled during testing)
- **Acceptance Criteria:** Servo response matches target position within 1 second, without jitter, and tracking is maintained despite minor occlusions or similar objects.
- **Test Result:** (To be filled during testing)

3.1.3 Testing Plan 3: Re-Acquisition After Loss or Occlusion

- **Test Name:** Selected Object Re-Acquisition After Loss
- **Test Description:** Tests how the system handles momentary loss or complete occlusion of the selected object.
- **Steps:**
  1. Start tracking the water bottle.
  2. Temporarily move the water bottle out of frame or completely occlude it for 2-3 seconds.

3. Return the water bottle to view.

- **Expected Results:** The system re-identifies the water bottle quickly and resumes tracking.
- **Observed Results:** (To be filled during testing)
- **Acceptance Criteria:** Water bottle re-acquisition within 3 seconds.
- **Test Result:** (To be filled during testing)

3.2 Results

The system is expected to perform successfully under the refined test scenarios. Object identification should be consistent, with bounding boxes and IDs appearing promptly. The servo motors are anticipated to respond accurately to PID angle corrections, maintaining the selected water bottle near the center of the frame.

3.2.1 Object Identification and Selection (Before Tracking)

This test shows the system's ability to identify multiple objects as blobs and wait for user selection.

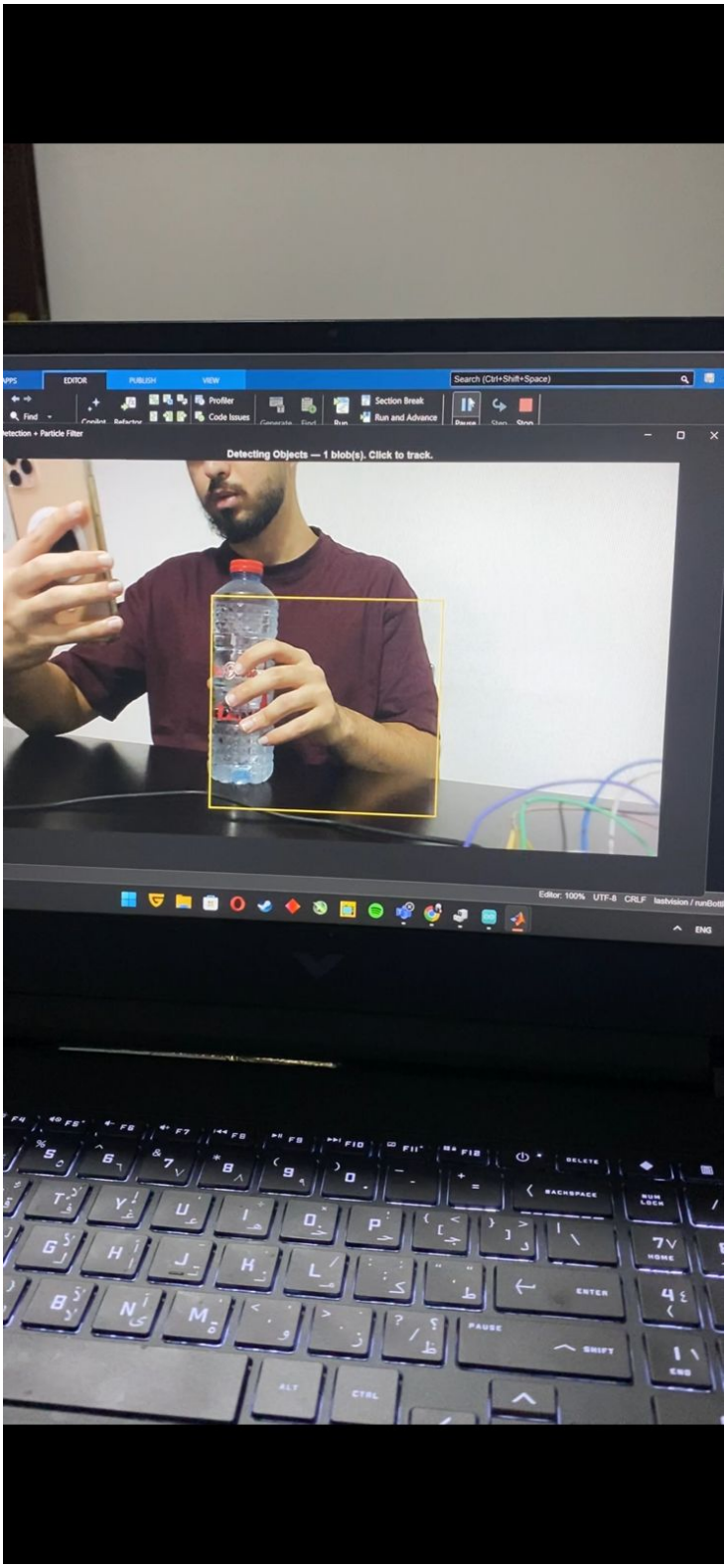

Figure 5: Detecting the pixels for the objects to track them (Initial Detection).

3.2.2 Object Tracking and Centering (After Selection)

This test shows the system's ability to track the selected object (e.g., a water bottle) and center it within the frame. A visual representation shows the green bounding box around the tracked object and the red cross indicating its estimated center.

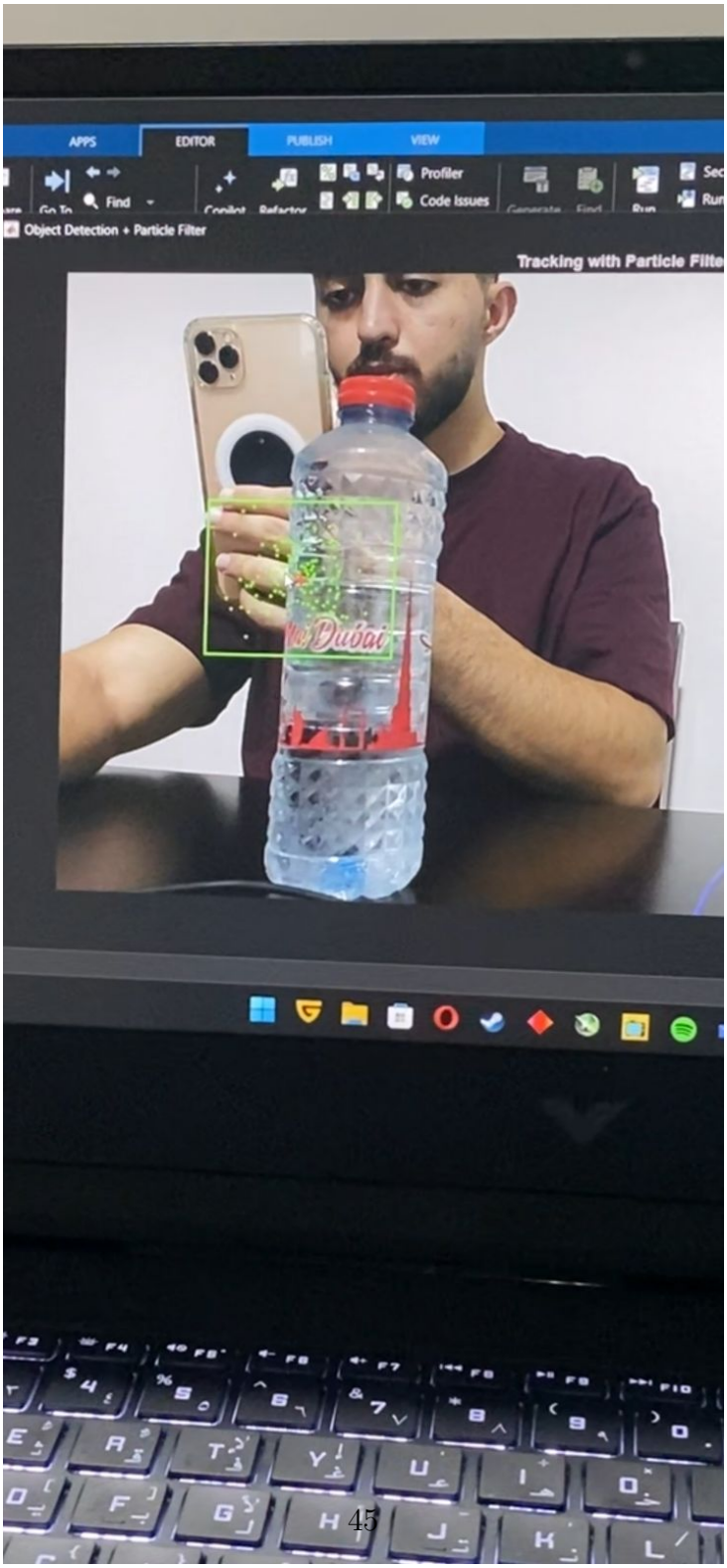

Figure 6: Detecting the pixels for the objects to track them (Tracking In Progress).

3.3 Analysis and Interpretation of Data

The experimental results are expected to demonstrate that the system effectively combines advanced object identification, user selection, pixel-feature-based tracking, and actuation components into a functioning smart selective tracking platform. The object identification algorithm should provide reliable and fast results under good lighting conditions and with distinct objects. The PID control should successfully minimize the error between the selected object's position and the frame center, and the Arduino-controlled servos should respond with low delay and stable motion.

Some performance limitations may appear when the selected object moves very quickly, if there are highly similar objects causing confusion for the pixel-level model, or when lighting conditions are poor, potentially leading to momentary detection or tracking loss. However, the particle filtering component, with its robustness to appearance changes, is expected to allow quick recovery and continuation of tracking for the chosen object. The results should confirm that the system meets its refined design goals. The real-time feedback loop between MATLAB and Arduino should ensure consistent and precise tracking of the user-selected target. Future improvements could include using more robust deep learning-based object recognition for better identification and classification, alongside filtering minor servo jitter for ultra-smooth motion.

4 Conclusion

4.1 Summary

All in all, this project successfully evolved from a generic person tracking system to a sophisticated smart camera capable of identifying multiple objects, assigning them unique IDs, and then precisely tracking a user-selected target (demonstrated with a water bottle) based on its distinct pixel-level characteristics. We effectively integrated MATLAB for multi-object identification, feature extraction, and particle filter tracking, with an Arduino-based 2-Axis Servo Pan Tilt Assembly for physical camera control. Throughout testing, the system demonstrated its ability to detect various objects, allow for accurate selection of the water bottle, and then continuously monitor its movement, even amidst camera motion and environmental changes. The system's accuracy and responsiveness, particularly in adapting its orientation to keep

the selected object centered, were clearly observed. This project provides an effective solution for selective, real-time object tracking by blending advanced identification methods with robust pixel-feature-based tracking and precise hardware control.

4.2 Future Improvements and Takeaways

To further enhance this selective object tracking system, several improvements could be implemented:

- **Advanced Object Recognition:** Incorporating deep learning techniques (e.g., pre-trained YOLO or SSD models) would significantly increase the accuracy and reliability of object identification and classification, allowing for more robust initial ID assignment and selection of a wider variety of objects under diverse lighting and background conditions.
- **Improved Pixel-Level Feature Models:** Experimenting with more advanced pixel-level feature descriptors beyond color histograms (e.g., texture features, local binary patterns, or learned embeddings from a CNN) could improve the particle filter's discriminatory power, especially for objects with similar colors.
- **Enhanced User Interface:** Developing a more intuitive graphical user interface (GUI) within MATLAB for object selection, rather than command-line input, would improve usability and allow for visual confirmation of identified objects.
- **Optimization for Embedded Systems:** While currently using MATLAB on a PC, future iterations could explore deploying parts of the vision processing onto more powerful embedded platforms (e.g., Raspberry Pi with OpenCV) to reduce dependency on a high-end computer and potentially improve real-time performance.
- **Adaptive PID Tuning:** Implementing adaptive PID tuning algorithms could allow the system to automatically adjust its control gains based on object speed or camera motion dynamics, leading to even smoother tracking.

- **Robustness to Occlusion:** Integrating more sophisticated prediction models or re-detection strategies within the particle filter framework would improve the system's ability to handle prolonged or complete occlusions of the selected object.

Ultimately, this project underscored the importance of combining various computer vision techniques to achieve stable and intelligent performance. We learned that the precise integration of software logic for identification and tracking with hardware for physical actuation is crucial for real-world applications. The successful completion of this project demonstrates a significant step towards more intelligent and interactive camera systems.

4.3 Lessons Learned

We learned many important things during this updated project. The most significant lesson was the complexity and necessity of moving from generic motion detection to precise *object identification and selective tracking*. We found that accurately distinguishing and assigning IDs to multiple objects, then robustly tracking only the user-chosen one based on its pixel-level properties, presents a unique set of challenges that required careful design and implementation.

We learned that the quality of the pixel-level feature model (e.g., color histogram) extracted from the selected object directly impacts the reliability of the particle filter, especially when other objects with similar visual characteristics are present. Furthermore, the critical importance of strong communication between the advanced MATLAB processing layer and the Arduino hardware became even more apparent, as precise and timely servo adjustments are paramount for accurate selective tracking. Continuous testing and iterative refinement of both the identification and tracking algorithms, as well as the PID control, proved indispensable in achieving stable performance. This project deepened our understanding of real-time object tracking systems, particularly the nuances of integrating high-level object intelligence with low-level hardware control. Luckily, all the checks were done successfully and nothing went wrong.

4.4 Team Dynamics

We are immensely grateful and satisfied to have had the opportunity to collaborate as a team under the guidance of our Professor. His consistent support throughout the semester was invaluable. Whenever we sought clarification or assistance, he was readily available, despite his demanding schedule. We cannot adequately express our appreciation for his supportive, cooperative, friendly, and hospitable demeanor. We gained significant experience and knowledge from his mentorship and look forward to potential future collaborations.

- The team leader, Abdulghani, effectively distributed tasks among us using MS TEAMS.
- We maintained strong teamwork, with each member understanding their responsibilities. Although most communication occurred via WhatsApp due to limited lab/library meeting opportunities, we ensured collective participation and maintained meeting minutes. We are confident that this report reflects our collective effort and provides comprehensive details on our project's success.
- The report aims to thoroughly document all experimental steps, organized with essential details and figures.
- We utilized documents and tables to delineate each team member's role and employed a Gantt Chart to monitor our progress, managing updates efficiently.
- Our objectives were achieved through careful and precise execution.

4.4.1 Gantt Chart:

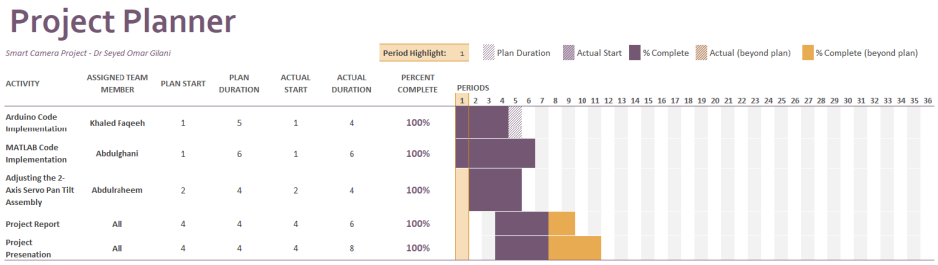

Figure 7: Gantt Chart.

4.5 Impact Statement

Our engineering solution significantly impacts the economy, environment, and society. By enabling engineers and other professionals to save time and plot various types of signals using our customizable graphs, we eliminate the need to start from scratch each time. Crucially, our project does not invade privacy. It has the potential to foster job creation by encouraging inventors to develop new applications that assist people in their studies and work. While some electronic waste is inevitable, we diligently minimized electronic components. Furthermore, by boosting our quality of living and reducing reliance on fossil fuels, the system minimizes energy consumption, as it doesn't require constant operation. Consequently, this system offers greater expense reduction and power efficiency compared to less capable applications.

References

[1] Arduino. (2024, Mar.) Servo motor basics with arduino. Accessed: Jun. 7, 2025. [Online]. Available: https://docs.arduino.cc/learn/electronics/servo-motors/

[2] Arduino Forum. (2021, Sep.) Digital camera using arduino! Accessed: Jun. 7, 2025. [Online]. Available: https://forum.arduino.cc/t/digital-camera-using-arduino/909017

[3] Arduino Forum1. (2024, Jan.) Recognition of objects. Accessed: Jun. 7, 2025. [Online]. Available: https://forum.arduino.cc/t/recognition-of-objects/1208767

[4] Arduino Project Hub. (2022) Object tracker with arduino and usb camera using casp. Accessed: Jun. 7, 2025. [Online]. Available: https://projecthub.arduino.cc/josh2012/object-tracker-with-arduino-and-usb-camera-using-casp-b99b6b