PRJ_371 MILESTONE 2

ROBOTICS GROUP IV

Karabo Linala (577593)
Kelo Letsoal (577613)
Johannes Gerhardus Van Wyk (578007)
Kamahelo Mototo (577715)
Jordan Miguel Barradas (577848)
Jamie Sepp Butow (577588)

Executive Summary

This report will detail Robotic Group IV's third year PRJ project milestone 2 which focuses on design and initial implementation. The report will first lay the groundwork of the project and elaborate in accordance with the selected software development methodology (scrum), the different sprints that will be undertaken during this milestone. These sprints, in accordance with scrum methodology and agile framework, will each follow several iterative steps, namely requirements gathering and analysis, system design, development and coding, testing and deployment / implementation. Written explanations, code snippets and other screenshots and diagrams will be provided with each sprint as well as a all updates to the sprint and project backlog.

Table of Contents

Executive Summary 1
Table of Figures 3
Introduction – Change in Hardware and Software for Project 4
Section A (Project Backlog & Selected Sprints for Milestone) 6
Section B – Sprint 1 (Robotic Arm Movement) 7
User Story 1 Design (GoToDesignatedSpot1) 8
User Story 2 Design (GoToDesignatedSpot2) 9
User Story 3 Design (GoToDesignatedSpot3) 10
User Story 4 Design (GoToDesignatedSpot4) 11
User Stories 1 to 4 Development / Implementation (Code Snippets) 12
User Stories 1 to 4 Testing (Sprint 1 Testing) 17
Section C – Sprint 2 (Phone API User Interface) 19
User Story 5 – Mobile App Control 20
User Story 6 – Object Detection Feedback 20
User Story 7 – Error Handling and Alerts 21
User Stories 5, 6, 7 – Notification User Interface Design 21
User Stories 5, 6, 7 – Home/ Main Page User Interface Design 22
User Stories 5,6,7 Psuedo Code Design 23
References 26

Table of Figures

Figure 1: Agile Software Development, 2024 (Source: Websmith Solutions, 2024: Online) 6
Figure 2: Robot Setup viewed from directly above 7
Figure 3: Location of User Story 1 within UML Diagram of RoboticArmMovement Class 8
Figure 4: Location of User Story 2 within UML Diagram of RoboticArmMovement Class 9
Figure 5: Location of User Story 3 within UML Diagram of RoboticArmMovement Class 10
Figure 6: Location of User Story 4 within UML Diagram of RoboticArmMovement Class 11
Figure 7: UML Diagram of RoboticArmMovementClass 12
Figure 8: Code Snippet Showing Declaration of Servo Objects used to Control Robot Arm 12
Figure 9: Code Snippet Showing setup() function 13
Figure 10: Code Snippet Showing loop() function 13
Figure 11: Code Snippet Showing moveLeft() function 14
Figure 12: Code Snippet Showing moveRight() function 14
Figure 13: Code Snippet showing moveToFowardPosition Function 15
Figure 14: Code Snippet showing moveToForwardPosition function 15
Figure 15: Code Snippet Showing reset Function 16
Figure 16: Code Snippet Showing pickUp() Function 16
Figure 17: Code Snippet Showing placeBack Function 17
Figure 18: Picture showing Robot Arm Successfully Picking up Object at Designated Spot 1 17
Figure 19: Picture showing Robotic Arm Place Down Object at Designated Spot 4 18
Figure 20: Picture showing Robotic Arm Successfully picking up Object from Designated Spot 3 18
Figure 21: Notification UI for Mobile App API 21
Figure 22: Image of "main" or home page of Mobile App 22

Introduction – Change in Hardware and Software for Project

The following will detail the current software and hardware components which will be used by this project in order to attain the goals set out in milestone 1 (the project execution plan).

Hardware Components

Main Source:

- ESP 32 Camera – For the object detection sensor

External Source:

- Raspberry Pico Servo drive – brain of the arm
- Six servos, or axes of rotation, give a robotic arm six degrees of freedom, which enables it to move in six different directions. Generally, these servos match the following:
    - Base servo – This is for the rotation; it controls the entire robotic arm around its base (left right)
    - Shoulder servo – Controls the movement of the upper arm. It moves the upper arm up and down just like how a person's shoulder rises at the joint and lower their arm
    - Elbow servo - Through bending at the "elbow" joint, the elbow servo gives the robotic arm the capacity to stretch or withdraw. This increases the arm's range of motion by enabling it to reach far or draw back.
    - Wrist rotation servo - The wrist rotation servo facilitates the rotation of the end effector, such as the gripper, around the forearm axis. This is crucial for changing the orientation of an object that the arm is holding or manipulating, such rotating a vertical object into a horizontal one.
    - Wrist tilt servo - Tilting the wrist up or down is made possible via the wrist tilt servo. This motion is comparable to how the human wrist can be tilted up or down. It makes it possible to handle and position items at various angles with greater accuracy.
    - Gripper servo - The gripper servo drives the end outcome, which is often a gripper used for object pickup. This servo allows the gripper to open, releasing things, or close, grasping them. The arm can now engage directly with the items it is intended to manipulate as this is the last phase of its activity.
- Five volts power supply

- Mounting Hardware: To put the robotic arm together and fasten the servo motors in position, a variety of screws, frames, and other mounting parts are needed/have been used.

Software Component

- C++
- Python
- OpenCV
- C#
- Ardruino IDE

The key components include the 6 DOF (Degree of Freedom) robotic arm, servo motors, a Raspberry Pico board, and the necessary power and control interfaces. Together, these components enable the precise movement and control of the robotic arm to perform tasks such as picking and placing objects in different positions.

The changes above in comparison to the original project execution plan were necessitated by the original robot arm selected for the project (KUKA) becoming unavailable for use. Because of this, changes to the original scope have become unavoidable: the creation of a separate user interface (mobile app Api) has become necessary as the project can no longer make use of the built in KUKA robot interface.

Section A (Project Backlog & Selected Sprints for Milestone)

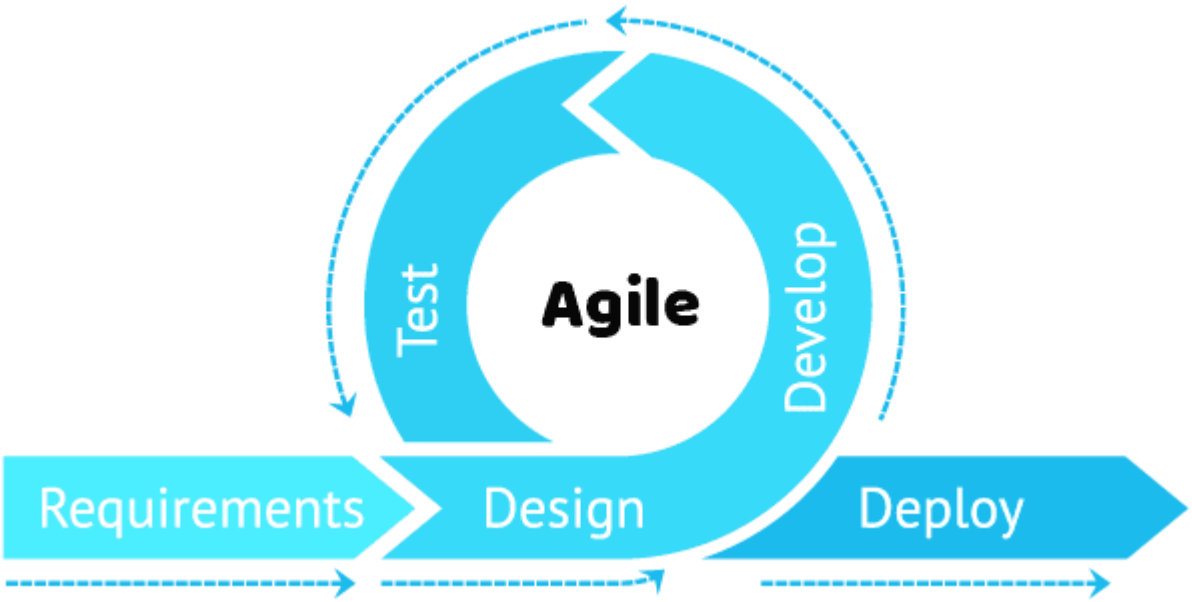


Figure 1: Agile Software Development, 2024 (Source: Websmith Solutions, 2024: Online)

In accordance with the selected agile software development methodology (scrum), this project will make use of the above steps in each sprint and or iteration. These sprints will consist of various user stories that will be made part of the selected sprint backlog for the sprints that will be implemented during this milestone. (Note: not all project backlog items will be implemented during this milestone, some will only be designed while others will be fully implemented and working). Selected sprints that will be tackled during this milestone and their planned degree of completeness are as follows:

- Robotic arm movement (to all 4 designated spots – movement up, movement down, movement left, movement right – in all 4 directions) – fully functional, tested and implemented (Achieved).
- API and dashboard to connect phone to allow for start/ stop of robot – designed, some code snippets but not yet fully implemented (Achieved).
- Object detection – designed, code snippets but not yet fully implemented (Not Achieved).

Sprint Name	Brief Description	Level of Completeness
(1) Robotic Arm Movement	Robot arm should be able to move to all designated locations in order to pick up and place down objects.	Fully designed, coded, tested, implemented.
(2) Phone API User Interface	User should be able to start stop sorting process via an app on their phone.	Designed, some pseudo code.
(3) Object Detection	Robot should be able to identify objects via a camera attached to arm.	Designed, some code.

Section B – Sprint 1 (Robotic Arm Movement)

Sprint Objective: the objective of this sprint is to get the robotic arm to move to 4 different designated spots in order to facilitate the picking up and dropping off of objects. This will be made use of during the robotic arm sorting process. The robot arm for demonstration purposes will sort 3 predetermined objects, the 4th spot will be made use of by the robot to allow for the sorting of the objects.

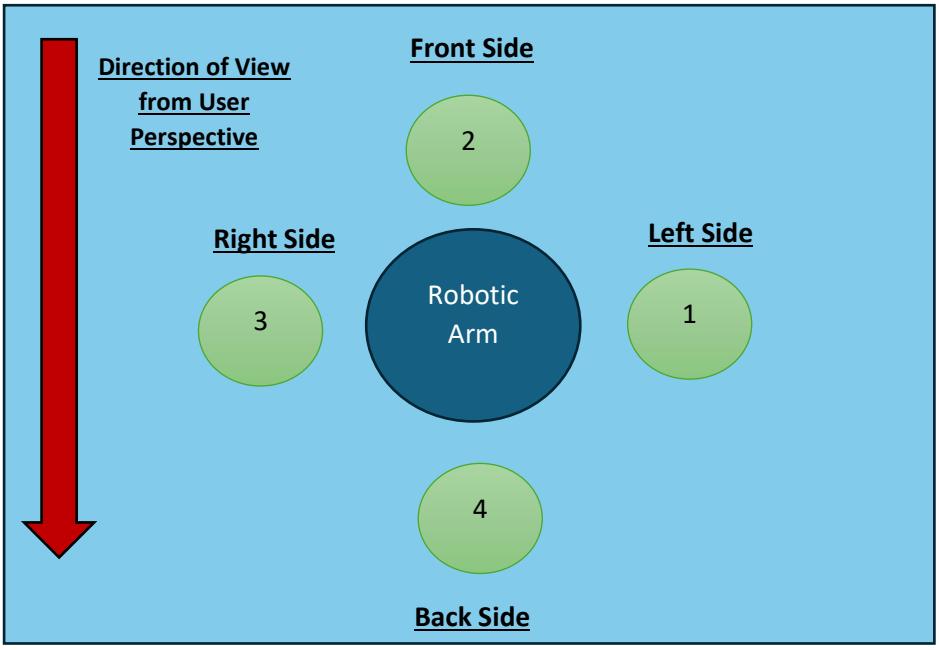


Figure 2: Robot Setup viewed from directly above

Key

- Predesignated picking up and dropping off spot. Spots 1 – 3 will house the items, while spot 4 will be used by the robot during the sorting process as a temporary holding spot for one of the items.

Sprint 1 (Robotic Movement) Sprint Backlog		
User Story Number	**Name**	**Description**
1	GoToDesignatedSpot1	Arm moves to designated spot 1 to pick up or place down objects
2	GoToDesignatedSpot2	Arm moves to designated spot 2 to pick up or place down objects
3	GoToDesignatedSpot3	Arm moves to designated spot 3 to pick up or place down objects
4	GoToDesignatedSpot4	Arm moves to designated spot 4 to pick up or place down objects

User Story 1 Design (GoToDesignatedSpot1)

User Story for User Story 1:

(Front of Card)

USER STORY TEMPLATE

USER STORY NUMBER: 1 **NAME:** GoToDesignatedSpot1

STORY: AS A USER, I WANT THE ROBOT TO BE ABLE TO GO TO DESIGNATED SPOT 1 SO THAT THE OBJECTS CAN BE PICKED UP AND PLACED DOWN

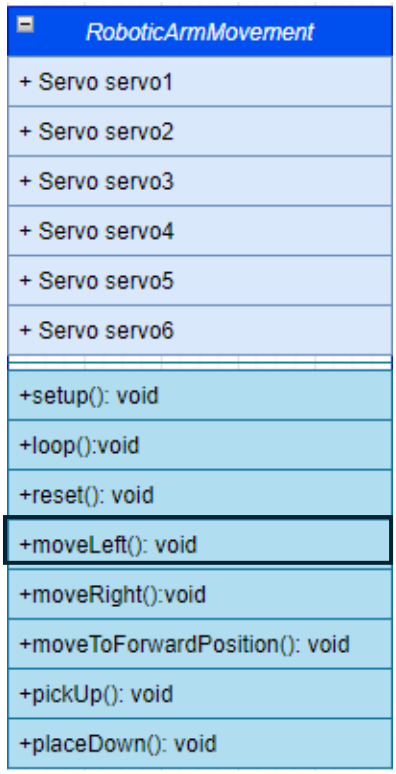
STATUS: ACCEPTED

(Back of Card)

USER STORY TEMPLATE

ACCEPTANCE CRITERIA:

ROBOT ARM SHOULD MOVE TO DESIGNATED SPOT 1 AND BE PERFECTLY ALIGNED TO PICK UP AND PUT DOWN OBJECTS AT THIS SPOT



This user story is implemented here. The various servos (motors) in the arm are manipulated to move the arm to the left position (to designated spot 1) – see diagram on page 5 for reference.

Figure 3: Location of User Story 1 within UML Diagram of RoboticArmMovement Class

User Story 2 Design (GoToDesignatedSpot2)

User Story for User Story 2:

(Front of Card)

USER STORY TEMPLATE

USER STORY NUMBER: 2 **NAME:** GoToDesignatedSpot2

STORY: AS A USER, I WANT THE ROBOT TO BE ABLE TO GO TO DESIGNATED SPOT 2 SO THAT THE OBJECTS CAN BE PICKED UP AND PLACED DOWN

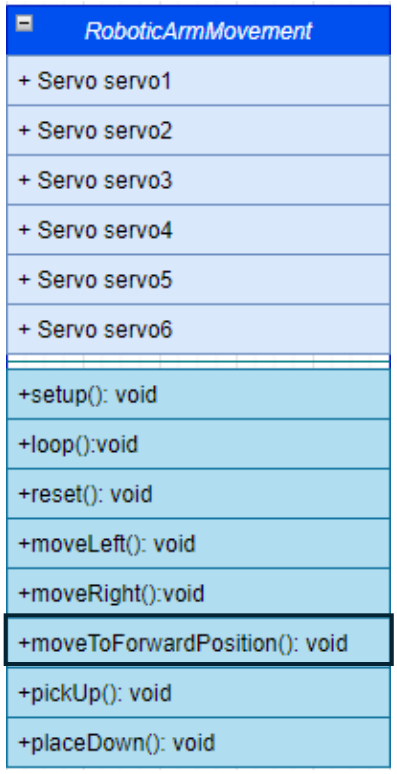
STATUS: ACCEPTED

(Back of Card)

USER STORY TEMPLATE

ACCEPTANCE CRITERIA:

ROBOT ARM SHOULD MOVE TO DESIGNATED SPOT 2 AND BE PERFECTLY ALIGNED TO PICK UP AND PUT DOWN OBJECTS AT THIS SPOT



This user story is implemented here. The various servos (motors) in the arm are manipulated to move the arm to the forward position (to designated spot 2) – see diagram on page 5 for reference.

Figure 4: Location of User Story 2 within UML Diagram of RoboticArmMovement Class

User Story 3 Design (GoToDesignatedSpot3)

User Story for User Story 3:

(Front of Card)

USER STORY TEMPLATE

USER STORY NUMBER: 3 **NAME:** GoToDesignatedSpot3

STORY: AS A USER, I WANT THE ROBOT TO BE ABLE TO GO TO DESIGNATED SPOT 3 SO THAT THE OBJECTS CAN BE PICKED UP AND PLACED DOWN

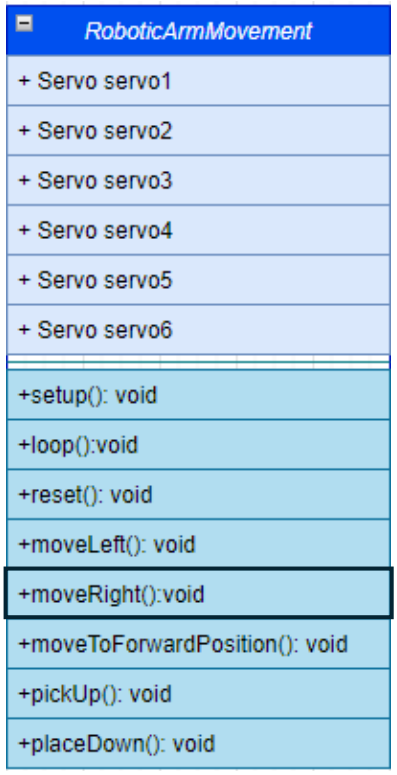
STATUS: ACCEPTED

(Back of Card)

USER STORY TEMPLATE

ACCEPTANCE CRITERIA:

ROBOT ARM SHOULD MOVE TO DESIGNATED SPOT 3 AND BE PERFECTLY ALIGNED TO PICK UP AND PUT DOWN OBJECTS AT THIS SPOT



This user story is implemented here. The various servos (motors) in the arm are manipulated to move the arm to the right position (to designated spot 3) – see diagram on page 5 for reference.

Figure 5: Location of User Story 3 within UML Diagram of RoboticArmMovement Class

User Story 4 Design (GoToDesignatedSpot4)

User Story for User Story 4:

(Front of Card)

USER STORY TEMPLATE

USER STORY NUMBER: 4 **NAME:** GoToDesignatedSpot4

STORY: AS A USER, I WANT THE ROBOT TO BE ABLE TO GO TO DESIGNATED SPOT 4 SO THAT THE OBJECTS CAN BE PICKED UP AND PLACED DOWN

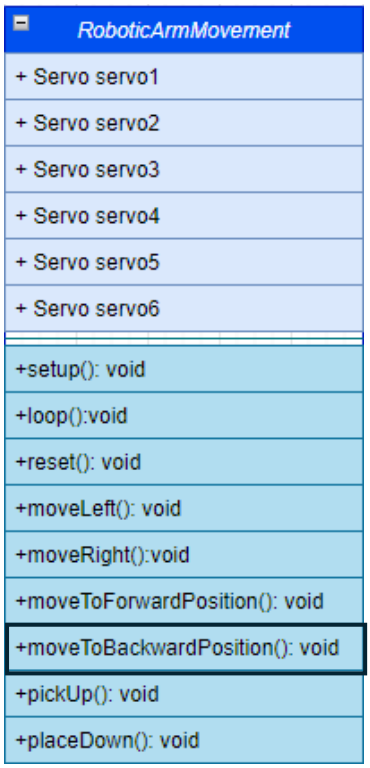
STATUS: ACCEPTED

(Back of Card)

USER STORY TEMPLATE

ACCEPTANCE CRITERIA:

ROBOT ARM SHOULD MOVE TO DESIGNATED SPOT 4 AND BE PERFECTLY ALIGNED TO PICK UP AND PUT DOWN OBJECTS AT THIS SPOT



This user story is implemented here. The various servos (motors) in the arm are manipulated to move the arm to the backward position (to designated spot 4) – see diagram on page 5 for reference.

Figure 6: Location of User Story 4 within UML Diagram of RoboticArmMovement Class

User Stories 1 to 4 Development / Implementation (Code Snippets)

In reference to the below UML diagram, this RoboticArmMovementClass consists of all necessary methods and attributes to implement User Stories 1 through to 4. Below are some code snippets with explanations where pertinent:

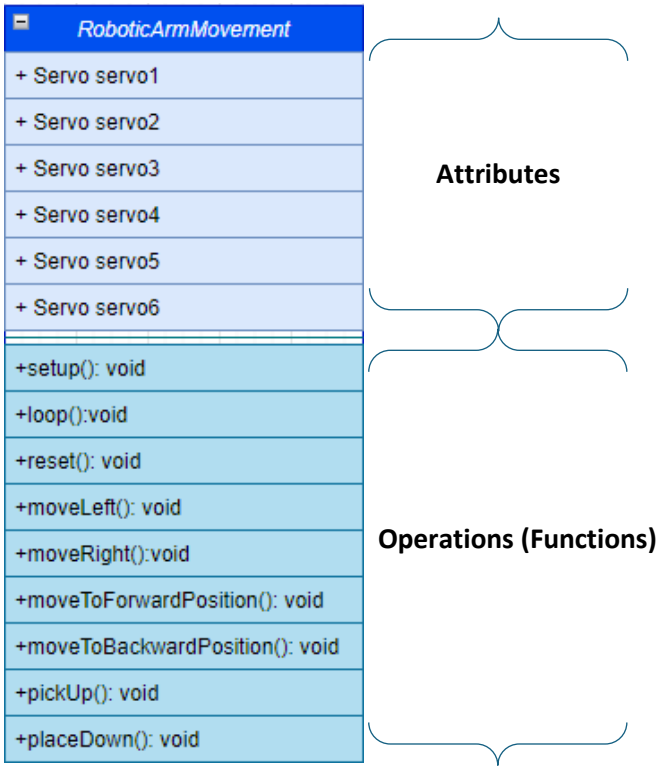


Figure 7: UML Diagram of RoboticArmMovementClass

Attributes:

Servo's 1 through to 6 are defined as Servo Objects and each reference a servo on the robotic arm. These are the motors that will be manipulated in order to make the robot move to each of the designated spots.

```
#include <Servo.h>

// Define servo objects for each joint of the robotic arm
Servo servo1;
Servo servo2;
Servo servo3;
Servo servo4;
Servo servo5;
Servo servo6;

// Define pin numbers for each servo

const int servo1Pin = 16;
const int servo2Pin = 17;
const int servo3Pin = 18;
const int servo4Pin = 19;
const int servo5Pin = 20;
const int servo6Pin = 21;
```

Figure 8: Code Snippet Showing Declaration of Servo Objects used to Control Robot Arm

Operations:

- Setup()

This method runs once and is used to attach the robots' servos to the servo objects declared above and also calls the moveToForwardPosition() function to ensure that the robot starts with the arm at designated point 1 (i.e. facing the user).

```
void setup() {

  // Attach each servo to its respective pin
  servo1.attach(servo1Pin);
  servo2.attach(servo2Pin);
  servo3.attach(servo3Pin);
  servo4.attach(servo4Pin);
  servo5.attach(servo5Pin);
  servo6.attach(servo6Pin);

  // Initialize all servos to the forward position
  // moveToForwardPosition();
  // delay(1000); // Wait for a second before starting movement
}
```

Figure 9: Code Snippet Showing setup() function

- Loop()

This method loops continuously and can be considered the "main" method of this class as this is where the order of movements is dictated. As of now, it commands the robot to move to each designated position.

```
void loop() {
  // Move the arm to the left
   moveLeft();
  delay(1000); // Wait for a second
  // Do pickup motion
  pickUp();
  delay(1000);
  // Do placing back motion
  placeBack();
  delay(1000);
  // Reset arm back to forward position
  reset();
  // Move arm back to right
  moveRight();
  delay(1000);
  reset();
  delay(1000)
  // Move arm back to backward position
  moveToBackwardPosition();

  // Move the arm back to the forward position
  moveToForwardPosition();
  delay(1000); // Wait for a second
}
```

Figure 10: Code Snippet Showing loop() function

- moveLeft()

Function implements the logic to make the arm move to designated spot 1. It does this by specifying the degree each servo should move from its current position.

```
void moveLeft() {
  // Adjust the angles to move the arm left
  delay(1500);
  servo1.write(160);  // Example angle to move left
  delay(3000);
  servo2.write(100);
  delay(3000);
  servo3.write(140);
  delay(3000);
  servo4.write(50);
  delay(3000);
  servo5.write(150);
  delay(3000);
  servo6.write(40);
  delay(5000);
  pickUp();
  servo6.write(90);
}
```

Figure 11: Code Snippet Showing moveLeft() function

- moveRight()

Function implements the logic to make the arm move to designated spot 3. It does this by specifying the degree each servo should move from its current position.

```
void moveRight() {
  // Adjust the angles to move the arm right
   delay(1500);
  servo1.write(10);
  delay(3000);
  servo2.write(100);
  delay(3000);
  servo3.write(140);
  delay(3000);
  servo4.write(50);
  delay(3000);
  servo5.write(150);
  delay(3000);
  servo6.write(40);
  delay(5000);
  pickUp();
  servo6.write(90);
  delay(1000);
  placeBack();

}
```

Figure 12: Code Snippet Showing moveRight() function

- moveToFowardPosition()

Function implements the logic to make the arm move to designated spot 2 (facing the user). It does this by specifying the degree each servo should move from its current position.

```
// void moveToForwardPosition() {
//   // Set all servos to the forward-facing position
//   servo1.write(90);  // Forward position
//   servo2.write(90);
//   servo3.write(90);
//   servo4.write(90);
//   servo5.write(90);
//   servo6.write(90);
// }
```

Figure 13: Code Snippet showing moveToFowardPosition Function

- moveToBackwardPosition()

Function implements the logic to make the arm move to designated spot 4. This spot will house an item temporarily during the sorting process. It does this by specifying the degree each servo should move from its current position.

```
void moveToForwardPosition() {
//   // Set all servos to the forward-facir
  delay(1500);
  servo1.write(90);  // Forward position
  delay(1500);
  servo2.write(90);
  delay(1500);
  servo3.write(0);
  delay(1500);
  servo4.write(60);
  delay(1500);
  servo5.write(90);
  delay(1500);
  servo6.write(90);
}
```

Figure 14: Code Snippet showing moveToForwardPosition function

- reset()

This is a helper function used to reset the position of the arm to designated position 2 (facing the user) as this position is used as a reference point for the other movement functions. For example, if the arm is currently at designated position 1 (left) and now needs to move to position 3 (right), the arm will first be set to position 2 (facing the user) using this function before function moveToRight() is called.

```
void reset() {
  servo1.write(90);
  servo2.write(90);
  servo3.write(90);
  servo4.write(90);
  servo5.write(90);
  servo6.write(90);
  }
```

Figure 15: Code Snippet Showing reset Function

- pickup()

Function simulates picking up of object (making robot arm grabber open and close down on an object). This is achieved by specifying the degree to which each servo should move from its current position.

```
void pickUp(){
  servo2.write(120);
  delay(1500);
   // when this goes down it hits no.4 before no.4 moves to 100
  servo4.write(100);
  delay(1500);
  servo3.write(160);// servo5.write(90);
  delay(1500);
  servo4.write(130);
  delay(1500);
  servo2.write(140);
  delay(2500);
  servo6.write(90);
//delay(500000);

  // servo6.write(90);
}
```

Figure 16: Code Snippet Showing pickUp() Function

- placeBack()

Function simulates placing down of an object my making the robotic arm's gripping mechanism relax its grip on the object (opens the pincers). This is achieved by specifying the degree to which each servo should move from its current position.

```
void placeBack(){
  servo2.write(100);
  delay(2500);
  servo1.write(90);
  delay(2500);
  servo3.write(140);
  delay(2500);
  servo4.write(80);
  delay(2500);
  servo6.write(60);
  delay(2500);
}
```

Figure 17: Code Snippet Showing placeBack Function

User Stories 1 to 4 Testing (Sprint 1 Testing)

In order to test the above code, the code was applied to the robot arm and was refined continuously until acceptance criteria of all 4 user stories was met. I.e., the robot moved to each designated position and the simulation of grabbing and dropping an object at each designated position was achieved. Attached below are pictures of the robot in action (a full video showing the entire result of the above sprint in action has also been included with the zip folder).

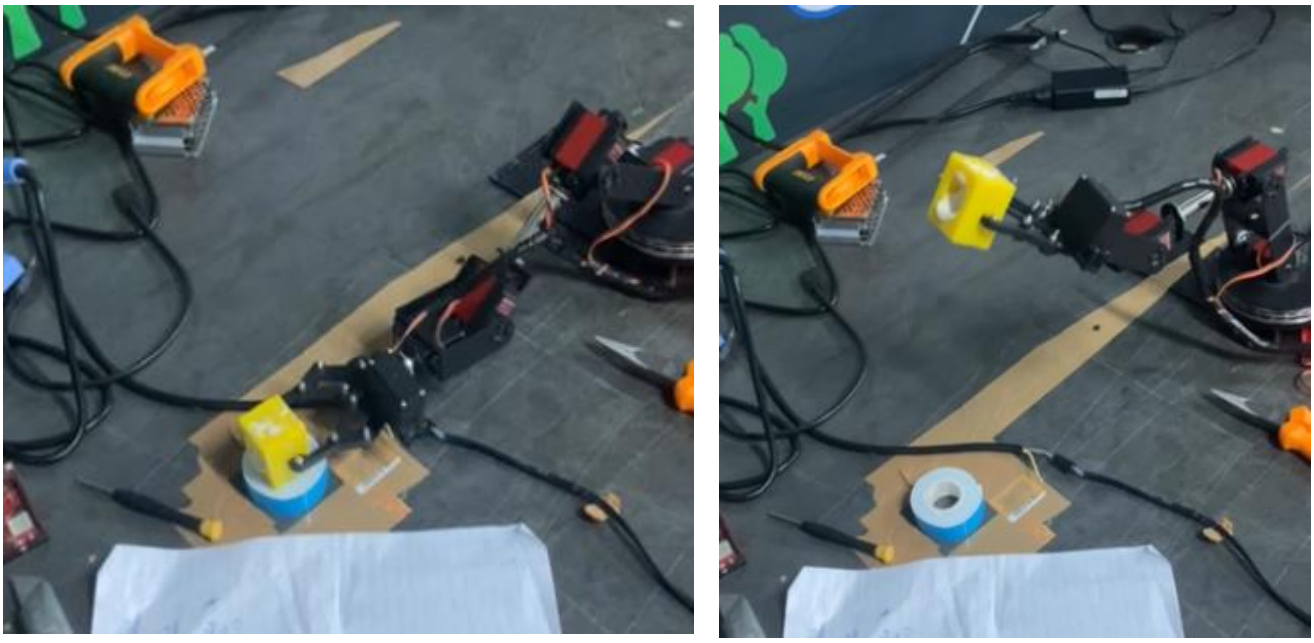


Figure 18: Picture showing Robot Arm Successfully Picking up Object at Designated Spot 1

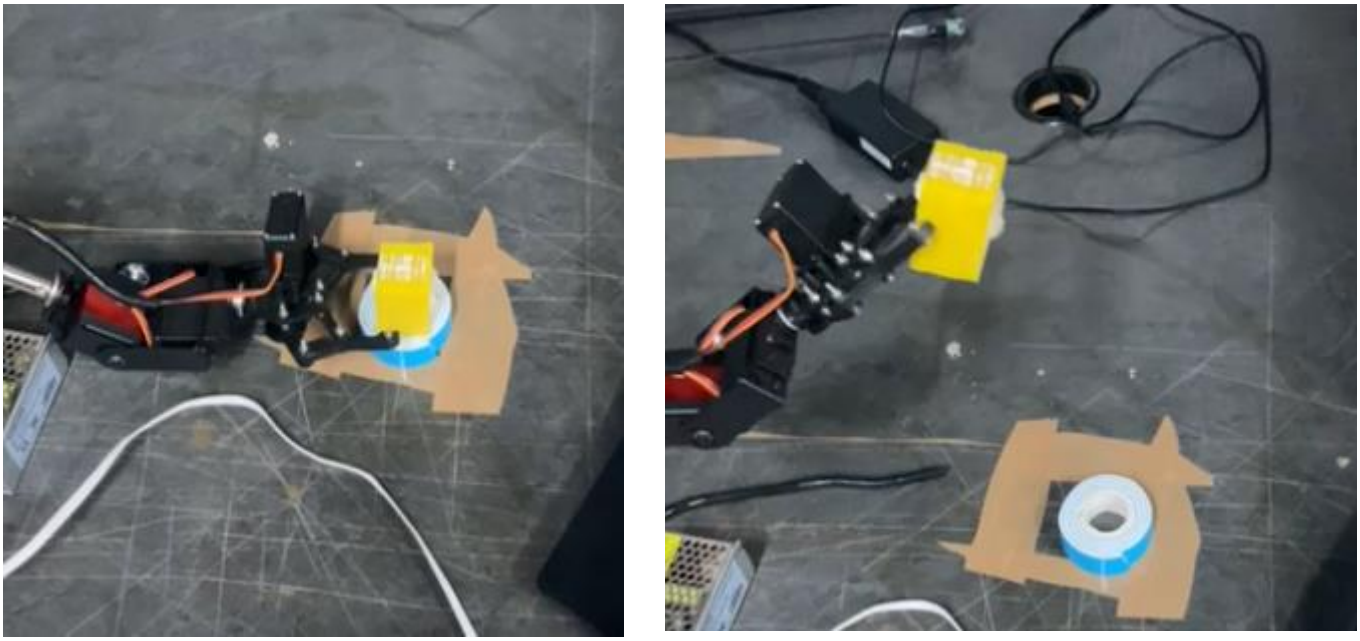


Figure 20: Picture showing Robotic Arm Successfully picking up Object from Designated Spot 3

Figure 19: Picture showing Robotic Arm Place Down Object at Designated Spot 4

Section C – Sprint 2 (Phone API User Interface)

The following will be implemented due to the change in hardware being utilized by this project. Namely, the change from the KUKA robot arm which came with its own interface to this new robot which does not have one.

Sprint objective: Develop and integrate the core functionalities of the mobile app and API to enable remote control, real-time feedback, and basic error handling for the robotic arm, ensuring a seamless user experience for monitoring and managing the sorting operations.

The following user stories will be incorporated into this sprint to allow the above sprint objective to be realized:

Sprint 2 (Phone API User Interface) Sprint Backlog		
User Story Number	**Name**	**Description**
5	**Mobile App Control**	Allows the sorting process to be started and stopped remotely through a mobile interface.
6	**Object Detection Feedback**	Allows users to receive alerts about object detection remotely through a mobile interface.
7	**Error Handling and Alerts**	Allows users to receive alerts about errors detected during the sorting process remotely though a mobile interface

The above user stories will be designed together as they are tightly integrated and apart of the same system.

The following section will elaborate on each of the above user stories and a design of the user interface that will incorporate all elements required to realize the objectives of the above user stories.

User Story 5 – Mobile App Control

USER STORY TEMPLATE

USER STORY NUMBER: 5 **NAME:** Mobile App Control

STORY: As a user, I want to use a mobile app to send start and stop commands to the robotic arm so that I can control its operation remotely

STATUS: ACCEPTED

USER STORY TEMPLATE

ACCEPTANCE CRITERIA:

The mobile app must allow users to send "Start" and "Stop" commands to the robotic arm, provide feedback on command status, and handle network errors gracefully

User Story 6 – Object Detection Feedback

USER STORY TEMPLATE

USER STORY NUMBER: 6 **NAME:** Object Detection Feedback

STORY: As a user, I want to receive real-time notifications on the mobile app about the objects detected by the robotic arm, so I can monitor the sorting process

STATUS: ACCEPTED

USER STORY TEMPLATE

ACCEPTANCE CRITERIA:

The mobile app must display real-time notifications of detected objects with relevant details and allow users to enable or disable notifications

User Story 7 – Error Handling and Alerts

USER STORY TEMPLATE	
USER STORY NUMBER: 7	**NAME:** Error Handling and Alerts
STORY:	As a user, I want to receive an alert on the mobile app if the robotic arm encounters an error or a jam, so I can take corrective action immediately
STATUS:	ACCEPTED

USER STORY TEMPLATE	
ACCEPTANCE CRITERIA:	The mobile app must display alerts for any robotic arm errors or jams, provide corrective action options, and log all alerts with timestamps

User Stories 5, 6, 7 – Notification User Interface Design

Notifications Design:

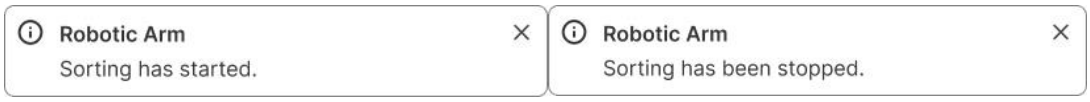


Figure 21: Notification UI for Mobile App API

The above design will facilitate user stories 6 and 7, alerting users of both errors and also of the state of the system as shown by the above image.

User Stories 5, 6, 7 – Home/ Main Page User Interface Design

The following will serve as the “main” or home page of the mobile app interface. Here the user can see the connection status of the app with the robot (connected or not connected), once connected can press the start button to start the sorting process, can click the stop button to stop the sorting process and under status can see messages relating to the sorting process (awaiting command, sorting has started, sorting has stopped):

Figure 22: Image of "main" or home page of Mobile App

User Stories 5,6,7 Psuedo Code Design

```
// Import necessary modules
Import express from 'express'

// Initialize Express application
Create an instance of Express: app

// Define the port for the server
Set PORT to 3000

// Middleware to parse JSON requests
Use middleware to parse JSON (app.use)

// Simulated data storage for status, notifications, and errors
Initialize roboticArmStatus to 'Stopped'
Initialize empty arrays for notifications and errors
```

```
// Define route for "Start" command
Define POST route at '/api/v1/robotic-arm/start'
    On request:
        - Try to send "Start" command to the robotic arm
        - Set roboticArmStatus to 'Running'
        - If successful, send JSON response { status: 'success', message: 'Robotic arm started successfully' }
        - If an error occurs, send JSON response { status: 'error', message: 'Failed to start the robotic arm' }
```

```
// Define route for "Stop" command
Define POST route at '/api/v1/robotic-arm/stop'
    On request:
        - Try to send "Stop" command to the robotic arm
        - Set roboticArmStatus to 'Stopped'
        - If successful, send JSON response { status: 'success', message: 'Robotic arm stopped successfully' }
        - If an error occurs, send JSON response { status: 'error', message: 'Failed to stop the robotic arm' }
```

```
// Define route for fetching object detection notifications
Define GET route at '/api/v1/robotic-arm/notifications'
    On request:
        - Fetch current notifications
        - Send JSON response { status: 'success', data: notifications }
        - If an error occurs, send JSON response { status: 'error', message: 'Failed to fetch notifications' }
```

```
// Define route for fetching errors and alerts
Define GET route at '/api/v1/robotic-arm/errors'
    On request:
        - Fetch current errors
        - Send JSON response { status: 'success', data: errors }
        - If an error occurs, send JSON response { status: 'error', message: 'Failed to fetch errors' }
```

```
// Define route for acknowledging errors or alerts
Define POST route at '/api/v1/robotic-arm/errors/acknowledge'
    On request:
        - Extract errorId from the request body
        - Search for errorId in the errors array
        - If found, remove the error and send JSON response { status: 'success', message: 'Error acknowledged' }
        - If not found, send JSON response { status: 'error', message: 'Error ID not found' }
        - If an error occurs, send JSON response { status: 'error', message: 'Failed to acknowledge error' }
```

```
// Define route for checking the robotic arm's current status
Define GET route at '/api/v1/robotic-arm/status'
    On request:
        - Return the current status of the robotic arm
        - Send JSON response { status: 'success', data: { currentStatus: roboticArmStatus } }
        - If an error occurs, send JSON response { status: 'error', message: 'Failed to fetch robotic arm status' }
```

```
// Start the server on the defined port
Listen on PORT
    On successful start, log message: 'Server running on http://localhost:PORT'
```

1. Connect the API and the mobile app using React and Flutter for HTTP requests and functions for each API call.

```
//Set up the main structure of the app, including necessary imports using dart.

import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: RoboticArmControl(),
    );
  }
}
class RoboticArmControl extends StatefulWidget {
  @override
  _RoboticArmControlState createState() => _RoboticArmControlState();
}

class _RoboticArmControlState extends State<RoboticArmControl> {
  String status = 'Awaiting command...';
  bool connectionStatus = false; // false for disconnected, true for connected
  final String apiBaseUrl = 'http://localhost:3000/api/v1/robotic-arm';
```

```
//Function to Send "Start" Command
Future<void> startRoboticArm() async {
  setState(() {
    status = 'Starting...';
  });
  try {
    final response = await http.post(Uri.parse('$apiBaseUrl/start'));
    if (response.statusCode == 200) {
      setState(() {
        status = json.decode(response.body)['message'];
        connectionStatus = true;
      });
    } else {
      setState(() {
        status = 'Failed to start the robotic arm.';
      });
    }
  } catch (error) {
    setState(() {
      status = 'Error starting robotic arm.';
    });
  }
}
```

```
//Function to Send "Stop" Command
Future<void> stopRoboticArm() async {
  setState(() {
    status = 'Stopping...';
  });
  try {
    final response = await http.post(Uri.parse('$apiBaseUrl/stop'));
    if (response.statusCode == 200) {
      setState(() {
        status = json.decode(response.body)['message'];
        connectionStatus = false;
      });
    } else {
      setState(() {
        status = 'Failed to stop the robotic arm.';
      });
    }
  } catch (error) {
    setState(() {
      status = 'Error stopping robotic arm.';
    });
  }
}
```

```
//Build the UI
@override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('PRJ ROBOTIC ARM')),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            // Connection Status Indicator
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: <Widget>[
                Text('Connection: '),
                Icon(
                  Icons.circle,
                  color: connectionStatus ? Colors.green : Colors.red,
                  size: 10,
                ),
              ],
            ),
            SizedBox(height: 20),

            // Start and Stop Buttons
            ElevatedButton(
              onPressed: startRoboticArm,
              child: Text('Start'),
              style: ElevatedButton.styleFrom(primary: Colors.green),
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: stopRoboticArm,
              child: Text('Stop'),
              style: ElevatedButton.styleFrom(primary: Colors.purple),
            ),
            SizedBox(height: 40),

            // Status Display
            Text('Status:'),
            Text(status, style: TextStyle(fontStyle: FontStyle.italic)),
            SizedBox(height: 40),

            // Bottom Navigation
            BottomNavigationBar(
              items: const <BottomNavigationBarItem>[
                BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Home'),
                BottomNavigationBarItem(icon: Icon(Icons.error), label: 'Error Log'),
              ],
            ),
          ],
        ),
      ),
    );
  }
}
```

References

Websmith Solutions. (2024) Agile Software Development. [Online Image][Accessed 29th August 2024]

https://websmithsolution.com/top-12-software-development-methodologies-its-advantages-disadvantages/