



**UNIVERSITY
OF MALAYA**



Software Requirement Specification

PaddyRobot: Automation for Material Transfer Using Robotic Arm – Prototype

Version: 2.0.0 Data: 28 January 2023

1 Table of Contents

2	Version Control	4
3	List of Abbreviations	5
4	List of Figures	6
5	List of Tables	7
6	Introduction	8
6.1	Background	8
6.2	System Overview.....	8
7	Project Overview.....	10
7.1	Team Overview	10
7.2	Aim and Objectives	11
7.2.1	Intended Audience and Intended Use	11
7.2.2	Product Scope	11
7.3	Constraints and Assumptions	12
8	User Persona and Characteristics	13
9	Hardware Requirements.....	15
10	Software Requirements	16
11	System Architecture.....	17
11.1	State Diagram.....	17
11.2	System Operation Flow Chart	18
12	External Interface Requirements	23
12.1	Graphical User Interfaces.....	23
12.1.1	User Management	23
12.1.2	Task Manager.....	24
12.1.3	Manual Control Mode.....	24
12.1.4	Automatic Control Mode	24
12.2	Hardware Interfaces	27
12.3	Communication Interfaces.....	27
13	Use Cases	32
13.1	Use Case Chart	32
14	Functional Requirements.....	34
15	Robotic Arm	37
15.1	Control Algorithm 1	38
15.2	Control Algorithm 2	40
16	Optimization	45

16.1	Operating Cost Optimization	45
16.2	Knapsack Algorithm	50
16.3	Travelling Salesman Problem	52
17	Summary	55
18	Authors Contribution	56
19	References	57
20	Appendix	58

2 Version Control

No	Version	Date	Description of changes and person responsible for making changes
1	0.1.0	06/12/2022	
2	0.2.0	09/12/2022	Update the functional requirement. (C.H. Chong) Update the system architecture. (C.H. Chong)
3	0.2.1	10/12/2022	Update captions, description of use case and one UI screenshot (Sara)
4	0.2.2	11/12/2022	Update section 6 Project Overview
5	1.2.2	11/12/2022	Submit to SE Team for Static Test Development.
6	1.2.3	17/12/2022	Updated use case diagram (Sara)
7	1.2.4	17/12/2022	Modified and updated the aim and objectives of the project. (C.H. Chong)
8	2.0.0	28/1/2023	Reformat the SRS document and retype some section. (C.H. Chong)
9	2.0.1	29/1/2023	Fix figure
10	2.0.2	30/1/2023	Added Keyboard control table

3 List of Abbreviations

No	Abbreviation	Description
1	PLC	Programmable Logical Devices
2	GUI	Graphical User Interface
3	SRS	Software Requirement Specification
4	DOF	Degree of Freedom

4 List of Figures

Figure 6-1: System Overview Diagram.....	9
Figure 7-1: Team Logo.....	10
Figure 8-1: End User Persona.....	13
Figure 11-1: System Architecture Diagram.....	17
Figure 11-2: System State Diagram.....	18
Figure 11-3: Startup Logical Flow.....	19
Figure 11-4: Shutdown Logical Flow	20
Figure 11-5: Trigger Start Logical Flow.....	21
Figure 11-6: Trigger Stop Logical Flow	22
Figure 12-1: GUI Screenshot.	23
Figure 12-2: Coordinate Calculation.	25
Figure 12-3: Coordinate Conversion Vector.	26
Figure 12-4: Coordinate Conversion Matrix.	26
Figure 12-5: Linear Conversion.	26
Figure 12-6: Cartesian diagram of coordinate translation.....	27
Figure 13-1: Use Case Diagram.	33
Figure 15-1: 6DOF Robotic Arm.	37
Figure 15-2: Robotic Arm Angles References.	38
Figure 15-3: State Diagram of Robotic Arm.....	39
Figure 16-1: Knapsack - Environment Setup.....	50
Figure 16-2: A snapshot of the production.....	51
Figure 16-3: Knapsack Implementation for Material Transfer.	52

5 List of Tables

Table 1 Team Members.	10
Table 2: User Access Level.	13
Table 3: Hardware Requirement List.	15
Table 4: Software Requirement List.....	16
Table 5: Functional Requirements.	34
Table 6: Knapsack Operation Result.	52

6 Introduction

This Software Requirement Specification (SRS) document outlines the software and hardware requirement for PaddyRobot project which uses robotic arm to assist paddy processing and packaging.

6.1 Background

The rice milling industry in Malaysia has been facing a multitude of problems that limits its efficiency and production output. Some of these are listed below. (Rice Processing, 2020)

1. Limited lands for the farming and setting up of the plantation for the paddy rice.
2. Low remuneration and unpromising career opportunities in the industry which results in limited labour force.
3. Low automation implementation for the industry be it within the plantation or rice milling process.
4. Under-utilisation of the existing rice milling by-product.

In this study, the focus is on the issue of low automation implementation for the industry. The low automation might be due to the reluctant adoption of the technology due habit of the older generation and the cost of the machinery. However, the specific factors contributing to the low adoption will not be explored.

The study will attempt to justify the use of robotic arm in the process of packaging with the use of optimisation for cost and performance of the technology implementation. In addition, the proof-of-concept for the robotic arm will be developed with the use of simple prototyping of code to setup a foundation for further research.

6.2 System Overview

The system is a design to model the material transfer process where goods or products are made from the manufacturing processes. The prototype or proof of concept for the system is proposed as shown below.

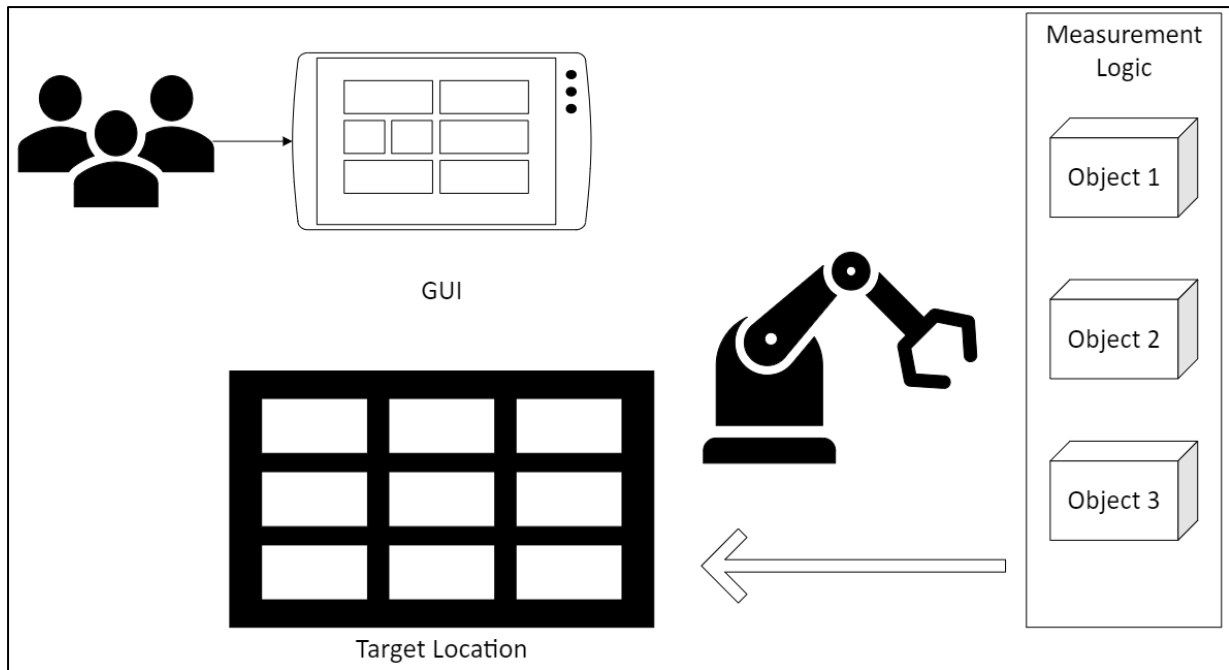


Figure 6-1: System Overview Diagram

The aim of the system is to move objects from a platform to another location to simulate the material transfer of the goods and products. The objects are chosen based on its weight to simulate the measurement process and transfer to a grid-like location where each grid can be specified from the software. The feature to pick specific location for transfer is to prototype the scalability and flexibility of the robotic arm in material transfer. Furthermore, the system should be fitted with a touch screen for end user operation on the system where basic trigger start or stop of operation can be done by operator.

7 Project Overview

This project is part of collaboration with ICT-INOV initiative which promotes “modernizing ICT education for harvesting innovation” by ERASMUS and Programme of European Union. Robotic arm usage within paddy processing has been identified as one of the areas to be studied.

7.1 Team Overview

There are 6 team members working on this project. The details are as shown.

Table 1 Team Members.

No	Name	Matric	Program
1	Chong Chia Hsing	S2159070/1	MCS AC
2	Lee Wai Key	17194567/2	MCS AC
3	Mohd Hafiidz Hassan	S2147238	MCS AC
4	Gary Yee Siew Wah	S2190954	MCS AC
5	Normaisarah binti Ab Majid	17150714	MCS AC
6	Yulin Lei	s2152115	MCS AC

The team logo is made of robotic arm and the team’s name PaddyRobot highlighting the focus towards the success of the project.



Figure 7-1: Team Logo.

The code and project information are available at the official project GitHub link as provided below. Do note that to ensure privacy, the project link is private, and access require admin approval. Please contact Hafiidz at S2147238@siswa.um.edu.my.

- GitHub link: <https://github.com/paddyrobot/woa7001>

7.2 Aim and Objectives

The aim of the project is to provide proof-of-concept for the automation of the material transfer process using the robotic arm which in turn can be applied in the context of rice milling industry.

The objectives of the projects are outline below:

- To build a prototype robotic arm which can perform the material transfer from 1 point to another location.
- To optimize the process of material transfer for rice milling factory in terms of the overall material cost, labour cost, power usage, and the average time to transfer.
- To optimize the robotic arm motion via suitable algorithm for more optimum usage in turn provide energy saving.

7.2.1 Intended Audience and Intended Use

The intended audience for the project is the managerial or rice milling factory owner as this project is to provide proof-of-concept for the implementation of robotic arm and justify its usage from economic perspective.

However, the end user of the system is targeted on the technician or engineer that will be operating or performing the packaging or material transfer process of the rice milling industry.

7.2.2 Product Scope

The project consists of both hardware and software prototyping and testing. The aim is to setup and construct a foundation for further investigation. The scopes are list as below.

1. Hardware prototype assembly
2. Software programming
 - a. Robotic arm control
 - b. Graphical user interface
3. Optimisation algorithms

4. Integration of hardware and software prototype

7.3 Constraints and Assumptions

The constraints that are faced in the project is that the material and equipment used for the prototyping. There is only 1 set of robotic arms for testing. Thus, testing on multiple set of robotic arms are not possible for optimisation algorithms.

In addition, due to there is several assumptions made on the condition and context of the environment that the robotic arm will be used. The robotic arm is assumed to be operating at the end of the production line where the rice will be packaged into a rice package and readied for transfer.

Furthermore, it is assumed that there are at least 3 manufacturing line to simulate a situation where optimization can be performed for the transferring the rice packages. These rice packages will also be queued at the end of the production line for the picking up by robotic arm as the manufacturing is in production.

8 User Persona and Characteristics

The user persona for the system is shown in the diagram below. The system will be designed with the stated users' characteristics in mind.

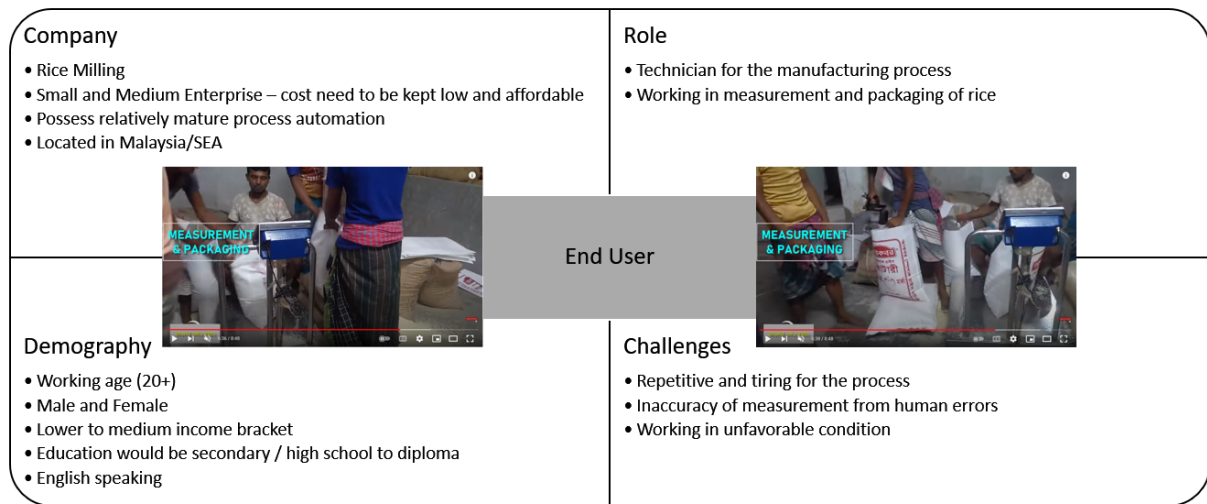


Figure 8-1: End User Persona

There are 3 possible users for the system which are the technician, engineer, and administrator. These 3 user types represent 3 different access level.

Table 2: User Access Level.

Actor/Role	Description	Required Knowledge
Technician	<p>Users can start and stop the robot arm.</p> <p>User can select features/routines from a list</p>	<p>Users need to know and understand the list of functions/routines of the robotic arm and can select and start/stop the robotic arm</p>

Engineer	<p>Users can perform all the Technician task.</p> <p>Users can change the configuration or threshold of the robot.</p>	<p>Users need to possess the technical knowledge and understand the basic of programming for the robotic arm and the safety and risk associated with the configuration.</p>
Admin	<p>Users can perform all the Engineer tasks.</p> <p>Users can create, delete, and change the Users.</p>	<p>Admin has the developer level understanding of the system and can provide customization and adjustment of the system based on the situation.</p>

9 Hardware Requirements

The hardware that are used for the construction of the robotic arm prototype are listed in the table below.

Table 3: Hardware Requirement List.

Feature ID	Hardware Feature Requirements	Description
HR001	Electrical power supply.	UNI-T UTP1306S DC Power Supply.
HR002	6 DOF Robotic Arm Frame	The frame that holds all components of robotic arm.
HR003	Laptop (Host Computer)	To code and implement the GUI
HR004	Servo Motor	To be able to move/rotate robotic arm to specific angle and position as outlined by user or algorithm
HR005	Camera	To use OpenCV library for generating coordinates using computer vision.
HR006	Arduino UNO R3	Microcontroller to receive instructions from laptop (host computer) and relay the right signal to move servo motors
HR007	Breadboard	To connect the electrical wiring for the robotic arm.
HR008	G clamps	To hold the robotic arm in place.
HR009	USB to Serial Port Cable	To bridge communication between laptop (host computer) and the Arduino board.

10 Software Requirements

The software requirements section will list out all the components used or involved in the development and deployment of the software.

Table 4: Software Requirement List.

No	Software	Description
1	Operating System	Microsoft Windows Environment
2	Arduino IDE 1.8.0	Software based on the C language used to program the code to be integrated to the controller
3	Programming Language	Python, C#, and C++
4	Arduino Board Driver or Libraries	
5	Graphic User Interface	Windows Forms Application is developed within the .NET environment using the Visual Studio IDE for the GUI.
6	OpenCV	This module is used for the computer vision application in the GUI for coordinate determination.

11 System Architecture

The system architecture is shown in Figure 11-1 where the relationship between each component of the system is linked together with an arrow indicating the flow of data or signal from and to the Arduino board which is the central controller for the system.

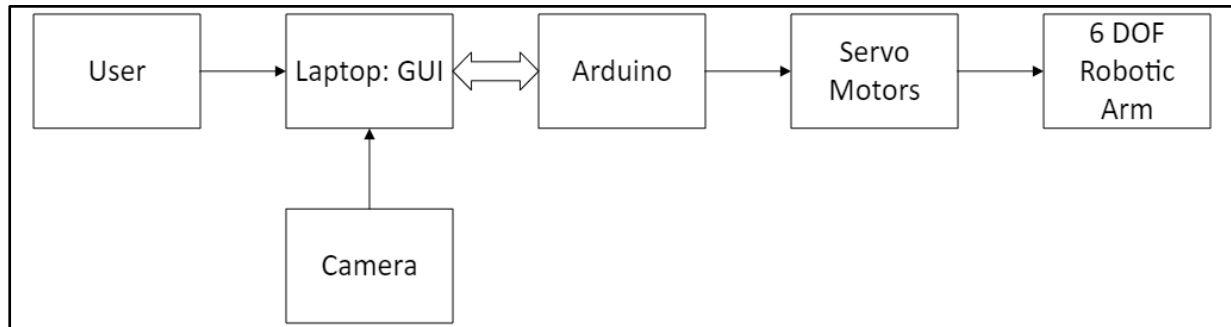


Figure 11-1: System Architecture Diagram.

It should be noted that the bidirectional arrow between the GUI and Arduino means that the data are pass to and from between these 2 components as the information of the current state of the system is stored in the Arduino and would need to be displayed to GUI while the input from user need to be pass from the GUI to the Arduino.

11.1 State Diagram

From the Figure 11-2, there are 4 possible states shown in circles for the system: (1) off, (2) on, (3) standby, and (4) operation. The arrows between these states are the transitions between states where the logical sequences need to be performed by the program to achieve the the respective states.

The “off” state is the situation where the system is powered down with no possible logical operation performed by the program. The “on” state represent the powered-up situation where there are sequence of initialization and checking performed but with no processes of operation by user or the robot. The external trigger and action is inhibit during this state.

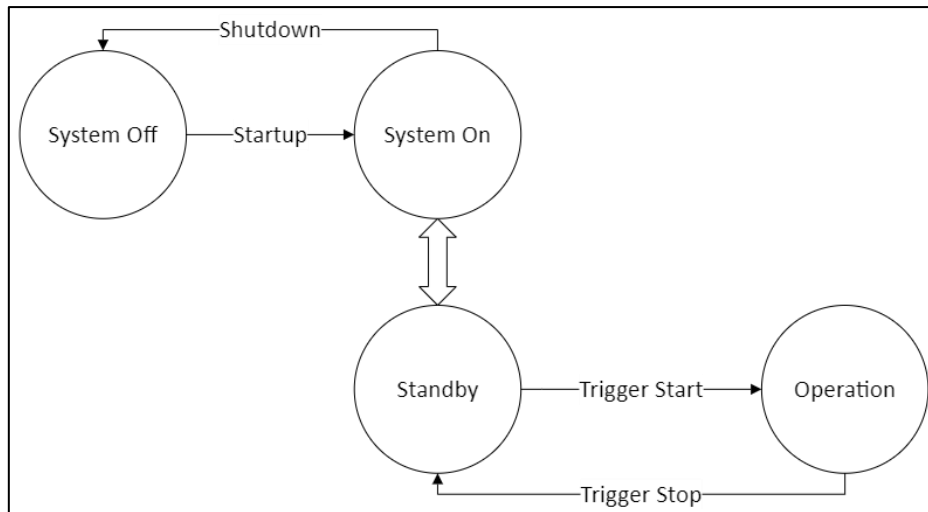


Figure 11-2: System State Diagram

The “standby” state is when the system is ready to perform robotic action based on any external triggers and condition set by design. Lastly, the “operation” state indicate that the robotic arm is in motion and the intended function of the system is being realized or performed.

The logical sequence to transfer the system state from “off” to “on” is termed “start-up” while “shutdown” is for the transition of “on” to “off”. On the other hand, “trigger start” and “trigger stop” is for transition of “standby” to “operation” and “operation” to “standby” respectively.

11.2 System Operation Flow Chart

The logical sequence for the transition between system states are shown as flow charts in Figure 11-3, Figure 11-4, Figure 11-5, and Figure 11-6. These figures correspond to the “Start-up”, “Shutdown”, “Trigger Start”, and “Trigger Stop” sequence respectively.

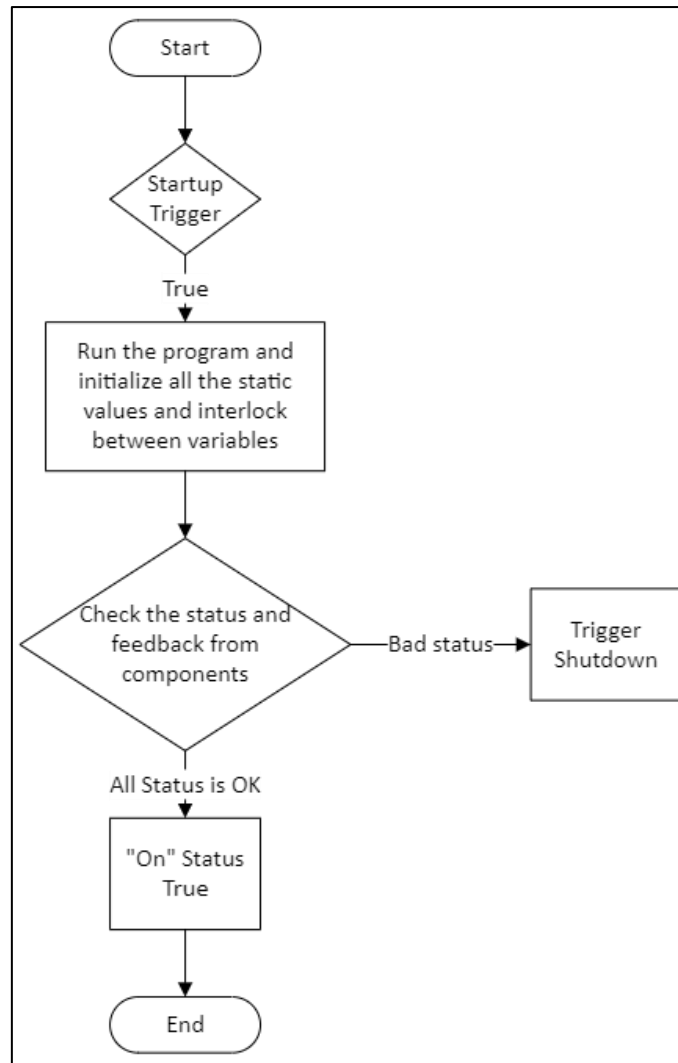


Figure 11-3: Startup Logical Flow

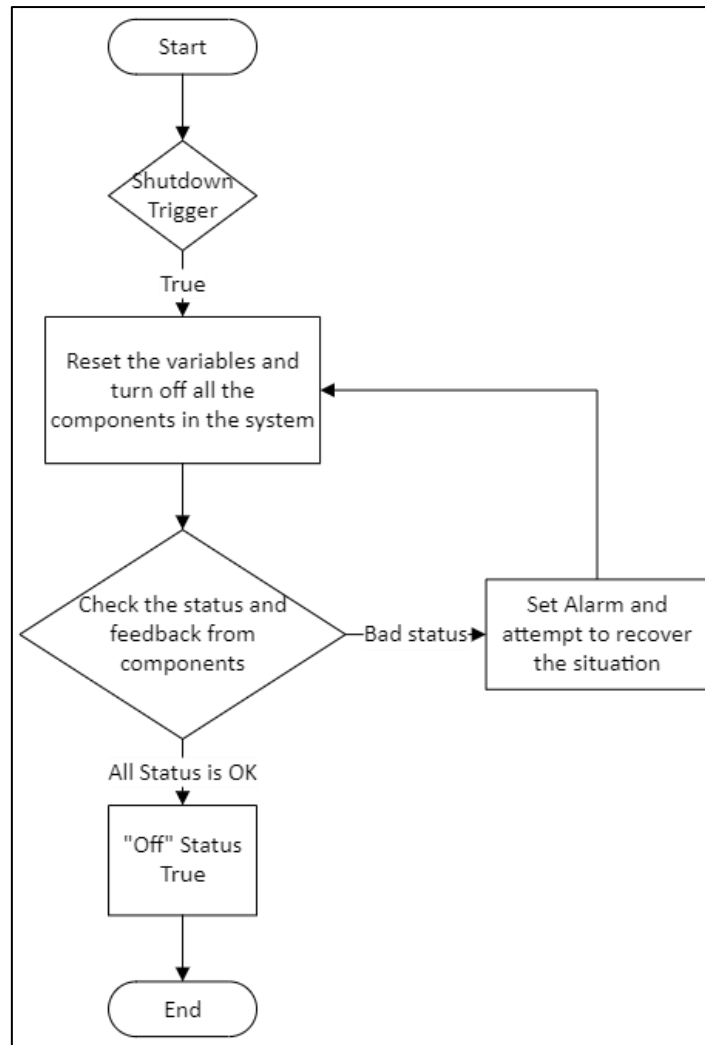


Figure 11-4: Shutdown Logical Flow

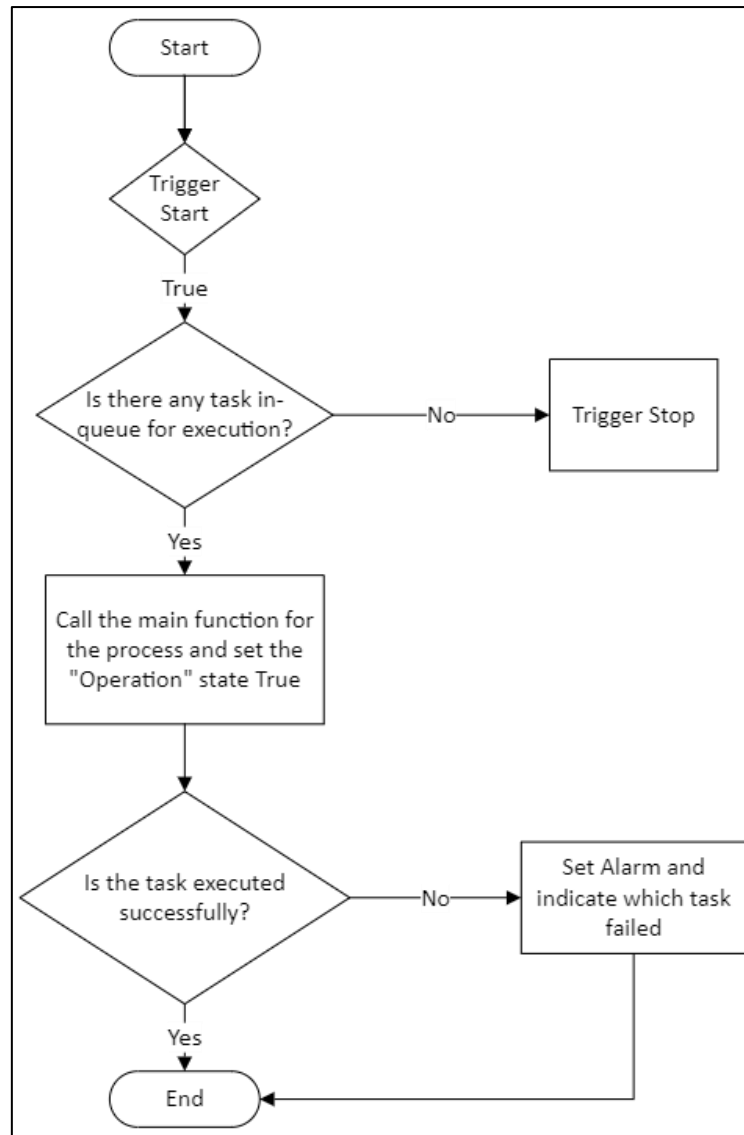


Figure 11-5: Trigger Start Logical Flow

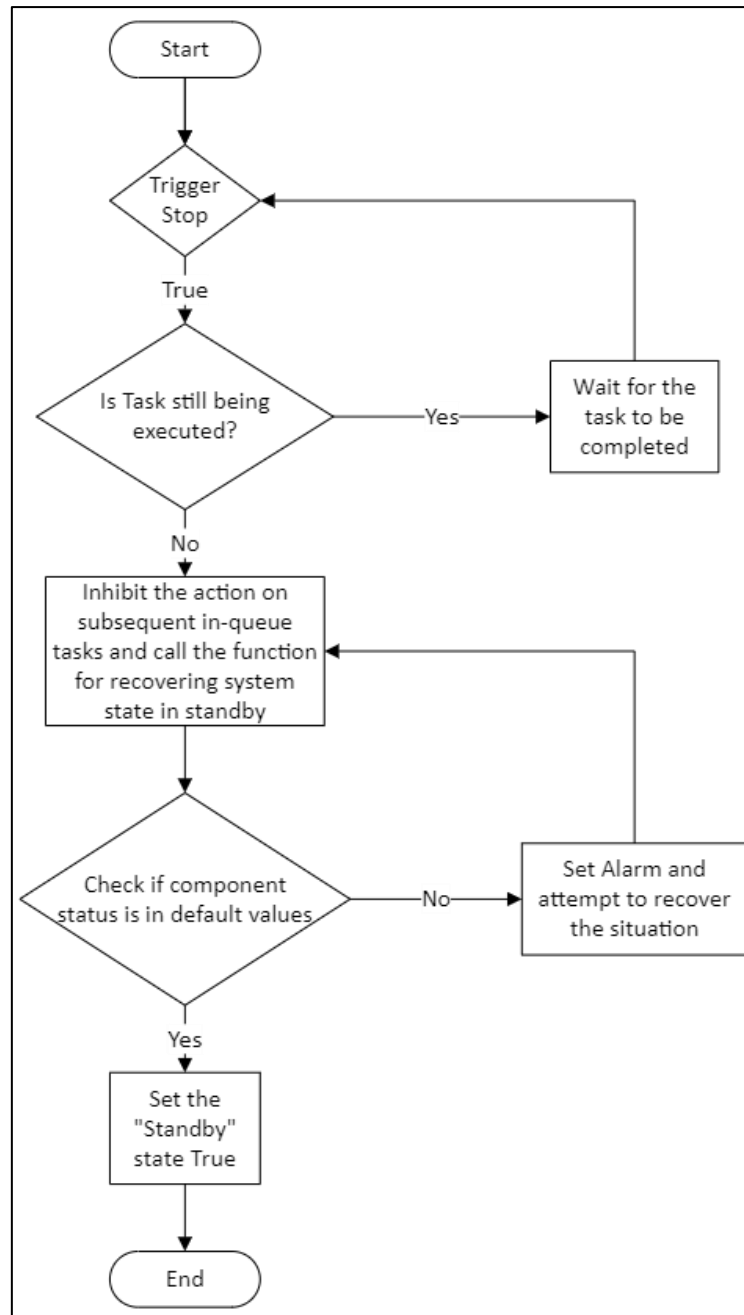


Figure 11-6: Trigger Stop Logical Flow

12 External Interface Requirements

This section will be discussing the interfacing done for the hardware, software, and with the user through graphical user interface (GUI). The purpose of interfacing is to perform information or data passing between the components within the system. The information can be passed as electrical signal (hardware), certain protocol of signal (software), or text semantic (user interface).

12.1 Graphical User Interfaces

The user interface is a Windows Forms Application, based on C#, running on the Windows system. It mainly has the following functions: 1. User management, 2. Task manager, 3. Manual debugging function, 4. Automatic control function. The above functions will be introduced respectively below. The GUI is shown in the Figure 12-1.

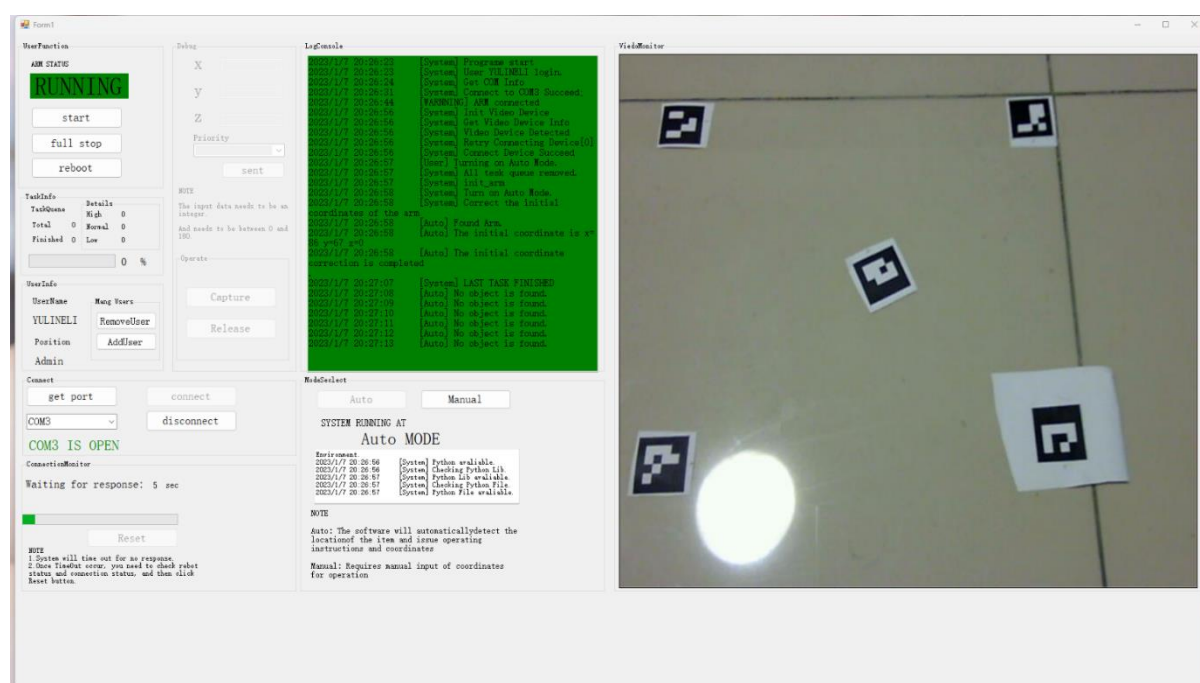


Figure 12-1: GUI Screenshot.

12.1.1 User Management

To ensure the safety of the machine operation, before entering the UI, the operator needs to login, and according to the role level of different users, they will also be given different functional permissions.

12.1.2 Task Manager

Tasks are managed using the queue data structure. This function can automatically send operation control instructions to the robot to realize operation automation.

12.1.3 Manual Control Mode

Users can manually operate the robotic arm according to their needs. The user needs to manually input the coordinates and click the send button, and the operation command will be sent to the robot arm for operation.

12.1.4 Automatic Control Mode

This mode is positioned by visually recognizing the Aruco mark. In this mode, the GUI can automatically recognize the object and convert the coordinates of the object into a coordinate address relative to the robot arm, and then send the coordinate address and operation instructions to the robot arm for execution. This allows the entire system to run autonomously without supervision. The flow chart of this mode is as follows:

1. Read the internal reference file of the lens.
2. Set the reference point information.
3. Specify the coordinates of the four-reference points mark.
4. Specify the markerID of the target.
5. Use the `aruco.detectMarkers()` function to detect the marker, and return the ID and the four corner coordinates of the marker board.

```
img_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
aruco_dict = aruco.Dictionary_get(aruco.DICT_4X4_50)
parameters = aruco.DetectorParameters_create()
```

6. With a set of 3D point coordinates in the physical world and 2D point coordinates in the image, and because the internal parameters of the camera are known, the R and T in the Figure 12-2 can be solved, that is, the transformation from the world coordinate system to the camera coordinate system relation.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

$$sP_i = A[R|T]P_w$$

Figure 12-2: Coordinate Calculation.

The `cv2.solvePnP()` function in OpenCV can solve the pose R and T of the camera based on a set of 3D-2D coordinates. `cv2.solvePnP()` does not directly return the rotation matrix R , but the rotation vector `rvec`. The two are directly equivalent, but the representation method is different. `cv2.Rodrigues()` is responsible for converting between the two formats.

```
retval, rvec, tvec = cv2.solvePnP(objectPoints, imagePoints, cameraMtx, dist)
rMatrix, jacobian = cv2.Rodrigues(rvec)
```

At this point, the external parameters of the camera are obtained, that is, the conversion relationship R and T from the world coordinate system to the camera coordinate system.

7. Also use `aruco.detectMarkers()` to detect the coordinates of the 4 corner points of the target marker in the image and find the mean value of the 4 points to calculate the coordinates (u, v) of the marker center point. The 3D points in the camera coordinate system are then calculated from the 2D points on the image. In fact, the 2D points on the image correspond to a series of 3D points in space, because the information of the depth z axis is unknown. According to the imaging model, the 3D points in the camera coordinate system can be calculated in this way. In addition to using the imaging model, the distortion of the lens needs to be considered. We can realize the above process through the `undistortPoints()` function in `Opencv`. Give the pixel coordinate `markerCenter` of the car target on the image and call it in the following way to get the coordinate point (x', y') whose z -axis is normalized to 1 in the camera coordinate system after correcting the distortion.

```
markerCenterIdeal=cv2.undistortPoints(markerCenter.reshape([1,-
1,2]),cameraMatrix,dist)
```

Add $z=1$, get the point $P1=(x', y', 1)$ in the camera coordinate system. $P1$ and the center point of the camera imaging point (that is, the origin of the camera coordinate system) $P0=(0,0,0)$ determine a ray, And the target is located on this ray. By calculating the intersection point of this ray with the floor plane, the coordinates of the target can be determined. First convert $P1$, $P0$ to the world coordinate system (Figure 12-3).

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

Figure 12-3: Coordinate Conversion Vector.

It can also be represented by the Figure 12-4Figure 12-1:

$$\begin{aligned} P_{camera} &= R \cdot P_{world} + T \\ P_{world} &= R^{-1} (P_{camera} - T) \end{aligned}$$

Figure 12-4: Coordinate Conversion Matrix.

According to the conversion relationship between the two coordinate systems, convert $P1$ and $P0$ to the world coordinate system. The straight line determined by these two points is expressed as Figure 12-5:

$$\frac{x - x_0}{x_0 - x_1} = \frac{y - y_0}{y_0 - y_1} = \frac{z - z_0}{z_0 - z_1}$$

Figure 12-5: Linear Conversion.

In the ground plane, $z=0$, put it into the above formula to calculate the position of the target on the motion plane ($x, y, z=0$). The Figure 12-6Figure 12-1 below shows the principle of planar object positioning.

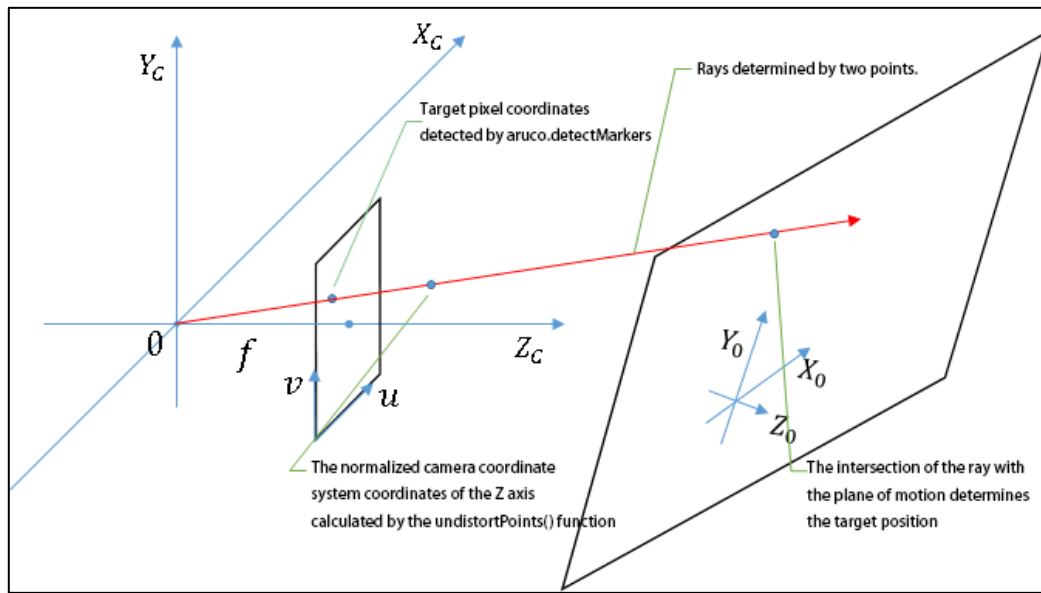


Figure 12-6: Cartesian diagram of coordinate translation.

12.2 Hardware Interfaces

The hardware interfaces consist of the cable and wiring that are used to power on and passes the electrical signal between the laptop and the Arduino board. The cable that is used to communicate with the Arduino controller board is an USB type A to USB type B cable.

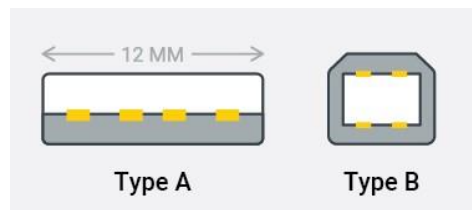


Figure 12-7: USB Type A & B

12.3 Communication Interfaces

There are two major communication interfaces in the robotic arm system. The first communication interface is between the laptop and Arduino controller board and the second is between the GUI and Python engine.

For the first interface, the Arduino board selected, Arduino UNO R3, is being programmed to use StandardFirmata library from the Sketch Library repository.

StandardFirmata library can be easily accessed from File> Examples > Firmata > StandardFirmata from within Arduino IDE as illustrated in the image below.

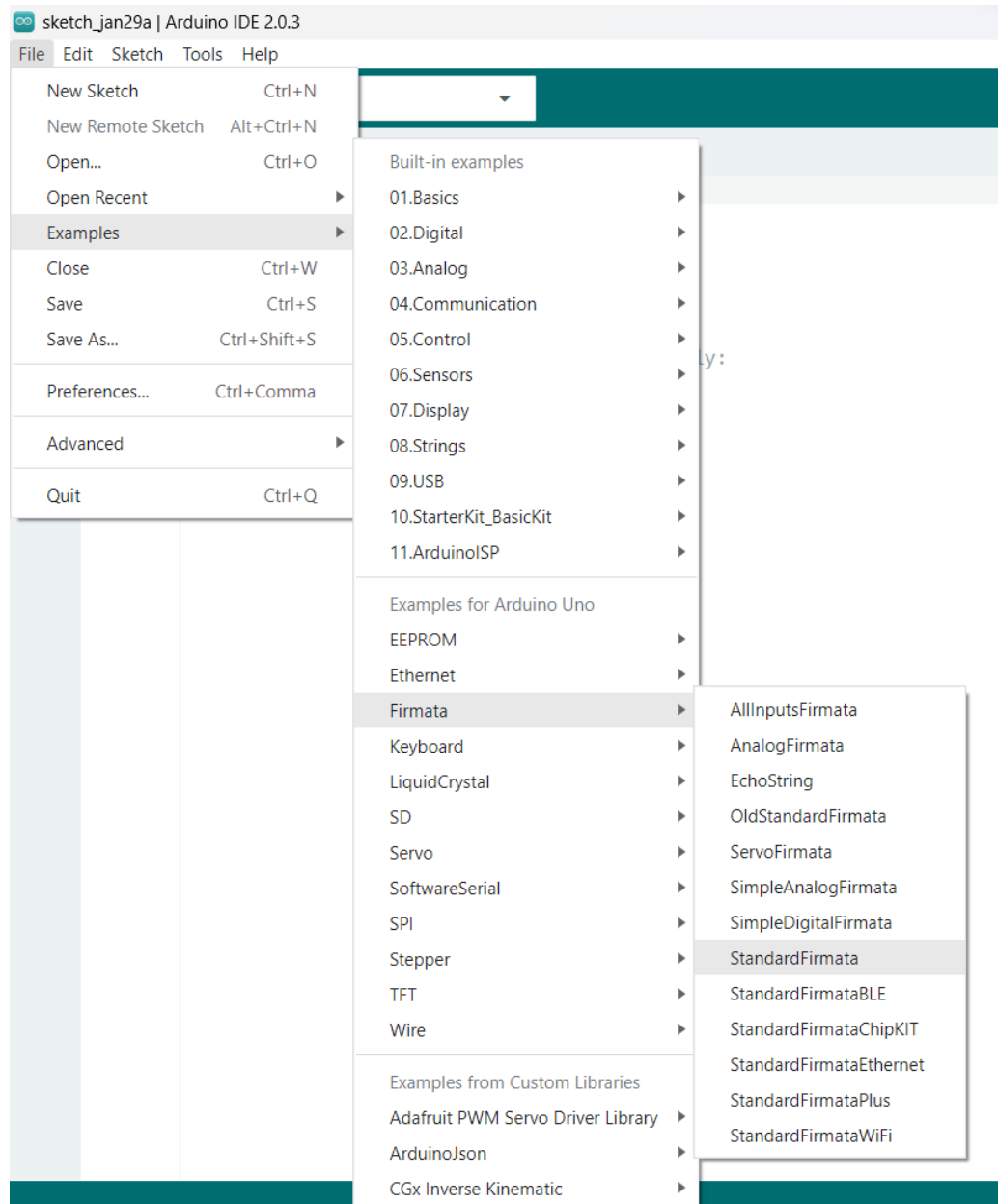


Figure 12-3: StandardFirmata Library in Arduino IDE

According to Firmata protocol definition web page written by Hoeffs et al (2022), Firmata protocol has been designed to allow for direct communication between microcontroller and a software object on the host computer and it was designed with the intention to allow for as much control as possible to be coming from the host system. In this particular application, the laptop used is the host, while Arduino UNO R3 board is the microcontroller that has been linked.

In the host laptop, we are using pyfirmata library to connect to the UNO R3 board. Some excerpts of the codes are shown below to illustrate the connection initiation and servos control.

```
1 > import pyfirmata...
8   filedir = path.dirname(path.realpath(__file__))
9
10  # select the port and assign the appropriate board, initiating the connection
11  port = 'COM3'
12  board = pyfirmata.Arduino(port)
13
14  # initiate servo
15  servo1 = board.get_pin('d:3:s')
16  servo2 = board.get_pin('d:5:s')
17  servo3 = board.get_pin('d:6:s')
18  servo4 = board.get_pin('d:9:s')
19  servo5 = board.get_pin('d:10:s')
20  servo6 = board.get_pin('d:11:s')

```

```
32  def write_pos(deg):
33      pos1, pos2, pos3, pos4, pos5, pos6 = deg[0], deg[1], deg[2], deg[3],
34      servo1.write(pos1)
35      servo2.write(pos2)
36      servo3.write(180-pos3)

```

An example from above, after we initiate servo by assigning the associated pin, for example, servo1 to pin #3 from UNO R3 board, we can directly control the signal from the pin to send a write signal to go to the right position, in this context, the angle at which the servo should go to in degree. More specifically, servo1.write(90) command would send a signal in Firmata protocol for the servo to move to 90-degree angle. In other words, we could send the necessary signal to the associated UNO R3 board pins to fully control the robot movement.

In addition, special setup has also been established to assign connection between specific keyboard press and the change in angle of the servo. For example, clicking the left button on the keyboard result in a decrease of 5 degree in servo1 value, and thus moving the

robot arm 5 degree to the left. In the similar but opposite direction, clicking the keyboard right button, increase the angle of servo1 by 5 degree. The relevant code is exhibited the next Figure 12-X.

```

141     # enable manual control, via keyboard
142     if(keyboard.is_pressed('m')):
143         manual = True
144
145     # enable auto mode, control via coordinate.json
146     if(keyboard.is_pressed('n')):
147         manual = False
148
149     # reset to default position, straight up
150     if(keyboard.is_pressed('r')):
151         pos1, pos2, pos3, pos4, pos5, pos6 = defaultpos
152         write_pos(defaultpos)
153
154         time.sleep(0.5)
155
156     if manual:
157         # drive left to right pos1
158         if(keyboard.is_pressed('right')):
159             if(pos1 + delta <= maxPos[0]):
160                 pos1 += delta
161                 servo1.write(pos1)
162                 time.sleep(0.5)
163         elif(keyboard.is_pressed('left')):
164             if(pos1 - delta >= minPos[0]):
165                 pos1 -= delta
166                 servo1.write(pos1)
167                 time.sleep(0.5)

```

On the other hand, between the Python engine and the GUI library, JSON data format is being used for communication interfaces. A simple example of this shown below.

```

codes > python > {} coordinate.json > ...
1  {"code": 1, "xyz": "150.0, -10.0, 100.0"}

```

The JSON file, consist of code and xyz key pair and an optional msg key. The code key value is tabulated in the next table.

Key Value Pair	Remarks
code: 0	Error code.
code: 1	Together with the xyz coordinate, move to the target location.
code: 2	Release. When this code is received, move the 6th servo (grip) to fully open and thus releasing object if any.
code: 3	Capture. When this code is received, move the 6th servo (grip) to fully close and thus capture object.
xyz: "x, y, z"	Contains the x, y and z coordinates in float data format.
msg: "string"	Contains optional msg string, used for debugging and other monitoring purposes

13 Use Cases

The use case here defined describes the system's operation in reference to the end users' operation. Furthermore, the specific nature of the use case is to provide an abstracted view on the operation of the system in which the primary function of the features can be deduced as it is stated with the end users' access level and operating steps.

13.1 Use Case Chart

There are 3 personas interacting with the system i.e., technician, engineer, and administrator. For basic tasks such as starting or shutting down the program and selecting tasks to be executed can be done by all personas.

^The engineer will have additional capabilities to update the configuration and settings such as updating the source code of the whole program, updating threshold and locations of the target starting point and destination.

An admin will have the same privileges as the technician and engineer but with an additional right to manage the users of the system. The admin will be able to add or delete users and assign appropriate roles to each of them.

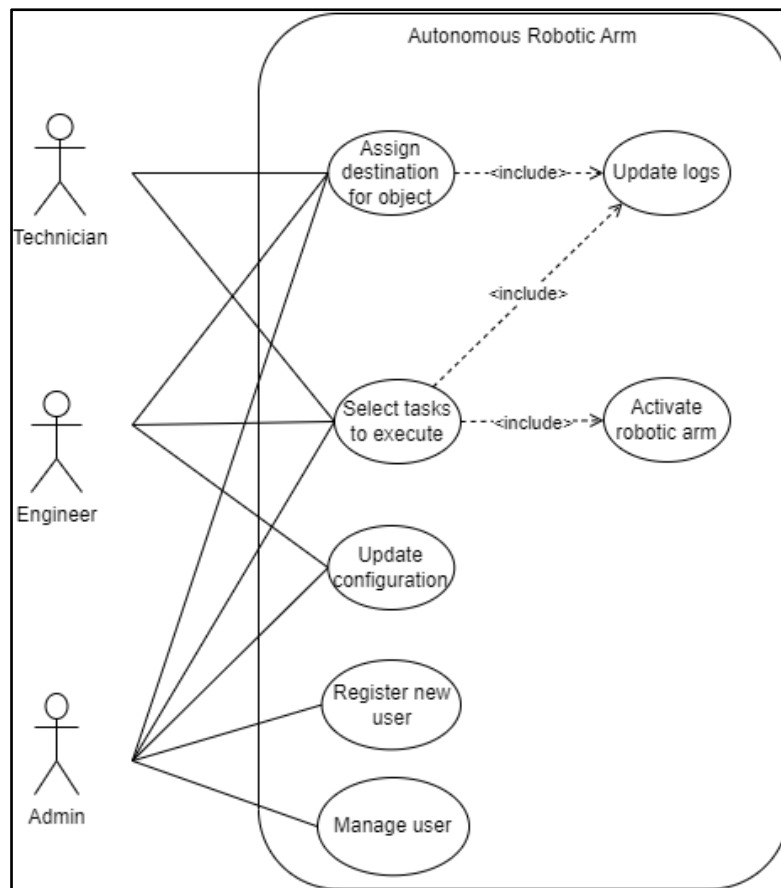


Figure 13-1: Use Case Diagram.

14 Functional Requirements

Functional requirements summarize all characteristics and features stated in the previous sections. These requirements range from system, user, hardware to software. Technical details might be included to help describe the functionality herein contained.

Table 5: Functional Requirements.

No	Requirements	Priority	Features
1	The system can power up all the components and devices.	High	Startup
2	The user interface is available and shows the statuses of the system and components.	High	
3	The system can perform some safety features and interlock to prevent mishandling of the system.	Medium	Safety
4	Set the maintenance mode for the system to disable the system ability to power on.	Low	
5	There are 3 roles or access level: technician, engineer, and admin.	High	User Access
6	The admin can create the user account and assign the different roles and possess full access to the system.	High	
7	The engineer can access and edit the configuration and threshold of the system and include the technician functionality.	High	
8	The technician can operate the system, e.g., startup, shutdown, create task, trigger start, and trigger stop.	High	
9	The user can create a task where a task is one complete operation of moving object from point A to B.	High	Task Creation
10	The user can decide the final location (point B) of the object for the task.	High	

11	The user can associate each different weight threshold for each task to decide the final location (point B) of the object.	High	
12	The creation of more than 1 task.	High	
13	The queueing of the tasks and first in first out in executing the tasks in queue.	Medium	
14	The display of the tasks in user interface.	High	Task Display
15	The deletion of the tasks in user interface.	Medium	
16	The timestamp for the tasks created.	Medium	
17	The timestamp for the tasks completed.	Low	
18	Trigger to start the system and execute the task.	High	Execution of task
19	Unable to trigger start the system if any of the components is in alarm or unhealthy mode.	Low	
20	Automatically execute the subsequent tasks in queue for multiple tasks.	Medium	
21	Manually trigger the start of the next tasks 1 at a time.	High	
22	Manually trigger the start of the next tasks 1 at a time even though there are multiple tasks in queue.	Medium	
23	Automatically stops when all the tasks in queue are completed or no task is in queue.	High	Stopping the execution of task
24	Trigger stop the system if any of the components is faulty during the operation.	Medium	
25	Manually trigger stop the system and the operation cease when the tasks in execution reach the end.	High	
26	Emergencies stop all the operation and power down the system.	Low	

27	The robotic arm can pick up the object based on the tasks created.	High	Operation of Robotic Arm
28	The robotic arm can transfer the object based on the tasks created.	High	
29	User able to manually operate the robotic arm.	Low	
30	Robotic arm pressure sensors detect the pressure exerted on the object.	Low	
31	The weight sensor can measure the weight of the objects.	High	Weight sensors for object detection
32	The system can decide the weight of the objects and place them in the designated location.	High	
33	The system can detect whether the destination location of the object is empty based on the weight sensors.	Low	
34	The system can decide the weight of the objects and place them in the most optimized location (nearest).	Low	

15 Robotic Arm

This section will provide further details and information on the robotic arm. The relevant specification on the robotic arm will be discussed from both software and hardware perspective along with the coding and program considerations.

The robotic arm used for prototyping is a standard prototyping 6 degree-of-freedom (6 DOF) robotic arm aluminium frame. It is built to be implemented with 6 servo motors to achieve the 6 motions of translation and rotation between the x, y, and z axis.



Figure 15-1: 6DOF Robotic Arm.

During the assembly of the hardware, it is discovered that the servo motors need to be properly calibrated with reference to the structure of the frame. This will ease development process as proper references to the angles are defined and it is shown in Figure 15-2.

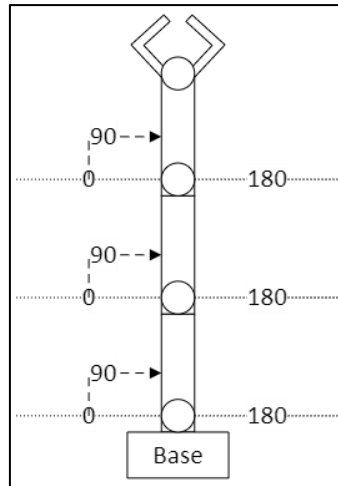


Figure 15-2: Robotic Arm Angles References.

Any persisting misalignment will be resolved with the use of offset variables in the Arduino code. In addition, the clockwise and anticlockwise control of the servo motor can be calibrated by multiplying the angle with negative value or using the equation.

$$Angle_{anticlockwise} = 180 - (Angle_{clockwise} + offset)$$

15.1 Control Algorithm 1

An initial control algorithm is proposed by Lei Yulin. The flow of the control logic is shown in the Figure 15-3.

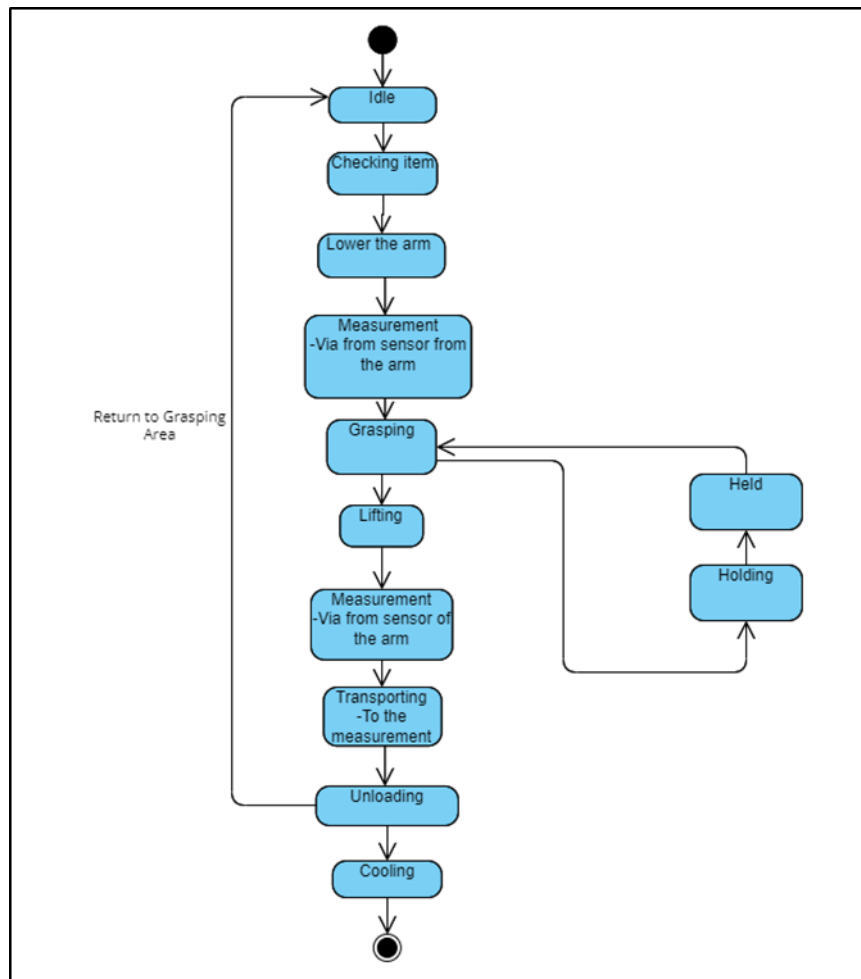


Figure 15-3: State Diagram of Robotic Arm.

At the start of the sequence, the machine is in an idle state. When the triggering condition for operation is fulfilled, the machine will move the robotic arm to the initial working position and point the camera at the assembly line.

After the alignment operation is completed, the camera will measure the position of the cargo. At this time, the robotic arm can obtain the offset between the cargo position and the preset position. Substitute the data into the algorithm to obtain a precise trajectory and grab the goods and start moving towards the pallet.

When the robotic arm moves to the preset position of the pallet, the camera will start to measure the cargo layer, and substitute the measured data to move the robotic arm to the top of the pallet. Then continue to make position corrections through the top cargo positioning point. Finally put the goods in the right place. Once the process is complete, the robotic arm can return to idle for the next job.

15.2 Control Algorithm 2

The robotic arm is controlled via the movement of six servos, precisely located at the required angles. The servos used in the robotic is often called as 180-degree servos, which has the characteristic of its ability to turn to an exact angle following the signal received from Pulse-Width Modulation (PWM) as opposed 360-degree servos which is continuous spinning servos, and can only be speed controlled.

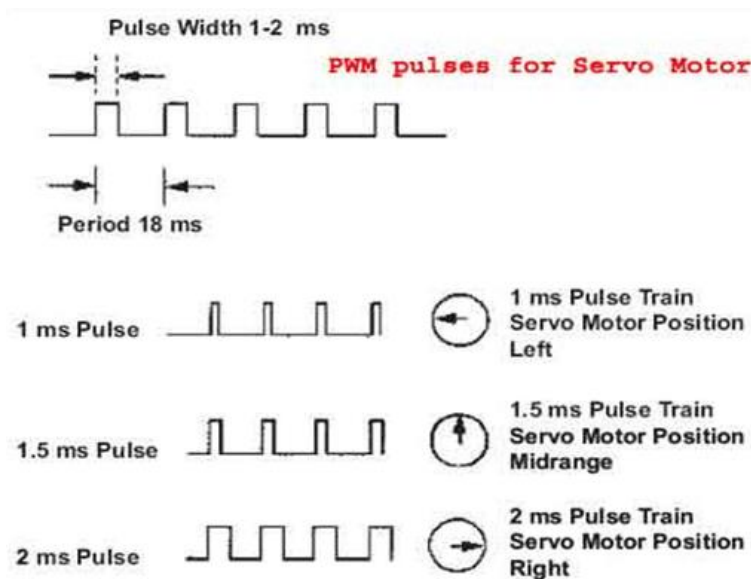


Figure 15-4: PWM and servo motor position (reference: circuitdigest.com).

PWM consists of repeating pulse that is send with a pre-determined length, height and frequency from the microcontroller board to the servo in order to communicate the exact angle or position for the servo to take. This allows for specific and direct control of the system. An example of this mechanism is illustrated in Figure 15-4 above, where different pulse signal will result in vastly different position. In practice, we have library in place to automatically convert the desired angle to the respective signal. As such, we can for example directly write the following code to reach desired angle.

```
23 # initiate servo
24 servo1 = board.get_pin('d:3:s')
25 angle = 60 # degree
26 servo1.write(angle)
```


In Arduino UNO R3, there are 6 pins that can interface using PWM mode, which are pin 3, 5, 6, 9, 10 and 11. Since we have a 6 degree of freedom robotic arm, there are 6 independent servos and as such, all 6 PWM pin on Arduino UNO R3 will be utilized. A simple circuit diagram for the servos and UNO board is illustrated below. Do note that there is also connection to pin 10 and 11 in addition to the 4 pin as per illustration.

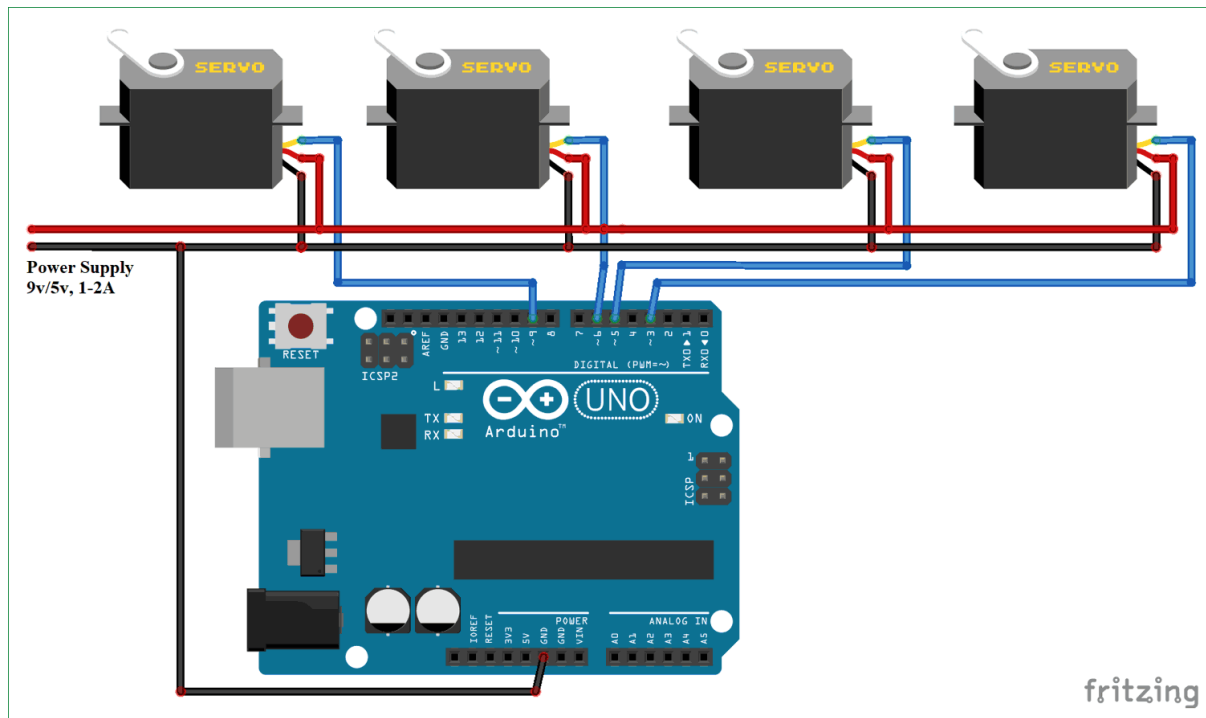


Figure 15-5: Diagram of UNO and Servo wiring connection (reference: circuitdigest.com).

Individually the servos require at least 5v power, which can be supplied from the UNO board via 5v power pin. However, when powering two or more servos, a higher voltage dedicated power supply is required. From our own testing, 7v is required for good functioning motors to reach the desired angle with sufficient force.

As briefly explain in section 12.3 Communication Interfaces, pyfirmata library is being used in the host computer to fully control the microcontrollers. All the calculation and optimization are being developed using Python language and instructions are sent via Firmata protocol to the Arduino UNO R3 board via Serial Com USB-A to USB-B cable. This instruction is later translated by the microcontroller to the respective servos via the signal pulse at designated frequency via the PWM pins and wired connection.

```

1 > import pyfirmata...
8   filedir = path.dirname(path.realpath(__file__))
9
10  # select the port and assign the appropriate board, initiating the connection
11  port = 'COM3'
12  board = pyfirmata.Arduino(port)
13
14  # initiate servo
15  servo1 = board.get_pin('d:3:s')
16  servo2 = board.get_pin('d:5:s')
17  servo3 = board.get_pin('d:6:s')
18  servo4 = board.get_pin('d:9:s')
19  servo5 = board.get_pin('d:10:s')
20  servo6 = board.get_pin('d:11:s')

```

```

32  def write_pos(deg):
33      pos1, pos2, pos3, pos4, pos5, pos6 = deg[0], deg[1], deg[2], deg[3],
34      servo1.write(pos1)
35      servo2.write(pos2)
36      servo3.write(180-pos3)

```

Figure 15-6: Code for pyfirmata, board & servos initiation as well as servos control.

Inverse Kinematics formula is being programmed with the help of pymoo library, which has a function to build a genetic algorithm to quickly solve the following algebra simultaneous equations given 4 unknowns, which is the degree of the angle of the servos with 3 given output variables, x, y and z which is the desired coordinates. x, y and z can be solved using function f1, f2 and f3 as outlined in the following codes. In principle they are actually trigonometry calculation based on the sin, cosine and hypotenus rules for a given length of each of the arm sections. As the angle decrease or increase will create the required motion to change the location of the gripper in respect to the base of the robot arm. Finally, there are additional control on the gripper position and direction, via the 5 and 6th servo angles. For example, at 90 degree, the gripper is in closed or captured position while at 30 degree, it is fully opened.

```

25 def f1(x):
26     deg0, deg1, deg2, deg3 = x[0], x[1], x[2], x[3]
27     return np.sin(np.deg2rad(180 - deg0))*(np.cos(
28         np.deg2rad(deg3)+np.deg2rad(deg2)+np.deg2rad(deg1)-np.deg2rad(180))*L3)+np.sin(
29         np.deg2rad(180 - deg0))*(np.cos(np.deg2rad(deg2)+np.deg2rad(deg1)-np.deg2rad(90))*L2)+np.sin(
30         np.deg2rad(180 - deg0))*(np.cos(np.deg2rad(deg1))*L1)+x0
31
32 def f2(x):
33     deg0, deg1, deg2, deg3 = x[0], x[1], x[2], x[3]
34     return np.cos(np.deg2rad(180 - deg0))*(np.cos(
35         np.deg2rad(deg3)+np.deg2rad(deg2)+np.deg2rad(deg1)-np.deg2rad(180))*L3)+np.cos(
36         np.deg2rad(180 - deg0))*(np.cos(np.deg2rad(deg2)+np.deg2rad(deg1)-np.deg2rad(90))*L2)+np.cos(
37         np.deg2rad(180 - deg0))*(np.cos(np.deg2rad(deg1))*L1)+y0
38
39 def f3(x):
40     deg0, deg1, deg2, deg3 = x[0], x[1], x[2], x[3]
41     return np.sin(np.deg2rad(deg3)+np.deg2rad(deg2)+np.deg2rad(
42         deg1)-np.deg2rad(180))*L3+np.sin(np.deg2rad(deg2)+np.deg2rad(deg1)-np.deg2rad(90))*L2+np.sin(
43         np.deg2rad(deg1))*L1+z0
44

```

```

5 # Using DE algorithm for more accuracy
6 from pymoo.core.problem import ElementwiseProblem
7 from pymoo.algorithms.soo.nonconvex.de import DE
8 from pymoo.operators.sampling.lhs import LHS
9 from pymoo.optimize import minimize

```

```

50 def solve(xyz):
51     # solve for xyz, as a tuple, (x, y, z ) target coordinate
52     # xt, yt, zt = (440, 0, 0)
53     xt, yt, zt = xyz
54
55
56 > class FindRadian(ElementwiseProblem): ...
57
58
59     problem = FindRadian()
60
61
62     algorithm = DE(
63         pop_size=100,
64         sampling=LHS(),
65         variant="DE/rand/1/bin",
66         CR=0.3,
67         dither="vector",
68         jitter=False
69     )
70
71
72
73
74
75
76
77
78
79
80
81 > res = minimize(problem, ...
82
83
84
85
86
87     X = res.X
88     F = res.F
89
90     X = np.round(X)
91
92     print("angles are = " + str(X))
93     print("final coordinates are = " + str(np.round((f1(X), f2(X), f3(X)))))
94
95     return (X, F)

```

Figure 15-7: Code for inverse kinematics, to calculate the optimum angle to reach the desired coordinates, x, y and z.

In addition to this, there is a manual mode to adjust the angle for each of the servo motors using keyboard press. For example, clicking 'r' button will result in system reset and move the robotic arm to position {0, 0, 400}. The detailed keyboard press and its associated action is included in the following table.

Key	Action	Remarks
R	Reset all position	Go to {0, 0, 400}, i.e. straight up
left ←	Reduce pos1 servo value by 1 delta (5 degree)	Turn left by 5 degree
right →	Increase pos1 servo value by 1 delta (5 degree)	Turn right by 5 degree
down ↓	Reduce pos2 servo value by 1 delta (5 degree)	
up ↑	Increase pos2 servo value by 1 delta (5 degree)	
S	Reduce pos3 servo value by 1 delta (5 degree)	
W	Increase pos3 servo value by 1 delta (5 degree)	
A	Reduce pos4 servo value by 1 delta (5 degree)	
D	Increase pos4 servo value by 1 delta (5 degree)	
Q	Reduce pos5 servo value by 1 delta (5 degree)	
E	Increase pos5 servo value by 1 delta (5 degree)	
X	Change pos6 value to 40	Release/open the gripper
C	Change pos6 value to 80	Catch/close the gripper
M	Change to manual mode	Might need multiple presses due to detection speed
N	Change to auto mode	Allow for the system to move according to pre-recorded movement sets from json file format

16 Optimization

This section will be discussing all the optimization algorithms and procedure that can be done to improve the cost and operating conditions of the system. The main purpose of the optimization is to justify the use of the system given from the cost optimizing perspective and to increase the efficiency and flexibility of the system.

16.1 Operating Cost Optimization

16.1.1 Variables definition

- i. The number of robotic arms, N - By increasing the number of robotic arms, we can handle higher throughput of rice packet transfer, which could ultimately reduce the overall cost. However, this will also increase the material cost of the robotic arms and the electric cost.
- ii. Robotic arm setup cost, C , in MYR – This is a static amount, one time of cost required to procure and setup one unit of robotic arm.
- iii. Throughput rate of the incoming rice packets, W , measured in packet per second - This is a random variable where in the production, the incoming rice packet rate varies from time to time, upper bounded by the upstream transfer rate (suppose this is 2 rice packets per second) and lower bounded by zero, when there are no incoming rice packets.
- iv. Material Transfer Time, T , measured in seconds – The amount of time required to transfer a rice packet, by a single unit of robotic arm. We assumed that the performance of the robotic arm can be controlled during the production phase (like an electric fan). With higher settings, better performance and hence lower material transfer time (however, the hosting cost will be higher due to higher electric consumption).
- v. Operation Cost, D , in MYR – Operation cost can be further divided into two components:
 - a. Maintenance cost is related to the deterioration of the robotic arm,
 - b. Hosting cost that is derived from the electricity consumption cost.

16.1.2 Determine the number of robotic arms to put into production

Before we put the robotic arms into production, we first need to determine the number of robotic arms required. The following is the details of the simple optimization problem:

Objective: Minimum total cost (Setup Cost + Operation Cost $\Rightarrow N * C + D$)

Constraint: The robotic arm(s) must be sufficient to support the incoming rice packets at any point in time (throughout the whole simulation period of 1 month)

Assumptions:

- Throughput rate of the incoming rice packets, $W \sim \text{Poisson}(1)$, average of 1 rice packets per second input rate as described in the previous section. A random sample is generated to represent the input rice packet rates, and a moving average of sliding window of 30 seconds is used to smoothen the input rate.

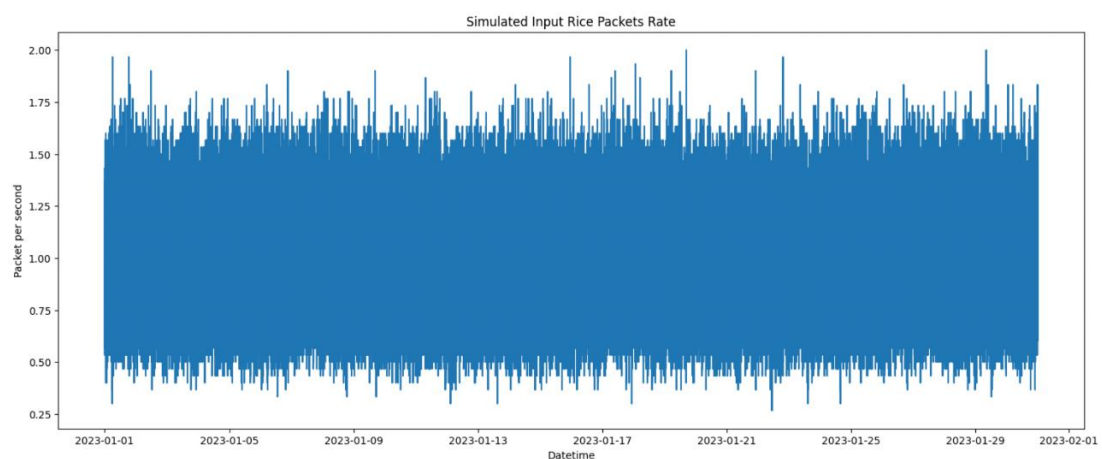
Here is the code to simulate the input rice packet rate

```
import numpy as np
import pandas as pd

input_rice_packet_rates = np.random.poisson(
    lam=1, size=int(30 * 24 * 3600) + 61
)
input_rice_packet_rates = pd.Series(
    input_rice_packet_rates
).rolling(30).mean()[60:].reset_index(drop=True)
input_rice_packet_rates
```

0	1.133333
1	1.066667
2	1.066667
3	1.066667
4	1.166667
...	...
2591996	1.366667
2591997	1.300000
2591998	1.300000
2591999	1.300000
2592000	1.266667

Following is the plot of the input rate



- There are 3 levels in the running settings of the robotic arms, described in the following table, where the material transfer time follows normal distribution for each level with different mean and standard deviation. In solving this problem, we further assume that the settings (level 1 to level 3) of the robotic arms are set fixed throughout of its lifetime.

Level	Average Material Transfer Time (seconds)	Std. Dev. Material Transfer Time (seconds)	Power Consumption (Watt)
1	12	1	20
2	9	0.6	35
3	6	0.3	75

To solve this problem, we will first observe the maximum input rates, which is about 2 rice packets per second in our simulation. In order to support this maximum rate, intuitively,

- If all the robotic arms are running in level 1 throughout the whole period, we will need about 24 robotic arms to achieve an average transfer rate of 2 rice packets per second. ($24 \times 1/12 = 2$)
- In another extreme case, if all the robotic arms are running in level 3 throughout the whole period, we will need only about 12 robotic arms to fulfil the demand. ($12 \times 1/6 = 2$)
- Therefore, the optimum number of robotic arms should lie between 10 to 20 robotic arms in this case.
- The main key point is the relationship between the material transfer time and power consumption is not linear, when the efficiency increases from 1 / 12 rice packet per second to 1 / 9 rice packets per second (by 1.33 times), the power consumption is increased by 1.75 times. Hence running in **lower speed** can **reduce our operation cost**. The tariff of 38.00 sen / kWh is used to simplify the problem.
- However, running in higher capacity requires lesser number of robotic arms, hence keeping the setup cost lower.

Next, we will form the objective function, the total monthly operation cost, M

$$M = N \cdot C + D(N, W(t))$$

N – Number of robotic arms

C – The setup cost of one robotic arm

W(t) - The input rice packet rate at time t

D – The annual operation cost, as a function of number of robotic arms and the input rice packet rate.

Next, we will run brute force search to generate multivariate normal samples for each N from 10 to 20 and, each (level1, level2, level3) combination within the fix N. We then calculate the total performance at any given time by summing up the random samples. Following is the code to

```
for N in range(10, 20):
    for a in tqdm(range(N)):
        for b in range(1, int(N - a)):
            total = (
                a / np.random.normal(12, 1, len(simulated_input))
                + b / np.random.normal(9, 0.6, len(simulated_input))
                + (N - a - b) / np.random.normal(6, 0.3, len(simulated_input))
            )
            simulated_dict[f"{N}_{a}_{b}"] = total
```

Only the combination with simulated total capacity higher than the simulated input rate with probability higher than 99.99% (< 0.01% of the time the robotic arms might not be able to support the input stream) is consider as success. The comparison is done in the following table by matching the input rate and capacity (both in the unit of rice packets per second) on each second.

	ts	input_rate	13_0_1_12	13_0_12_1
0	2023-01-01 00:00:00	1.133333	2.111333	1.534122
1	2023-01-01 00:00:01	1.066667	1.941855	1.469638
2	2023-01-01 00:00:02	1.066667	2.090082	1.565703
3	2023-01-01 00:00:03	1.066667	2.262191	1.544139
4	2023-01-01 00:00:04	1.166667	2.077752	1.432915
...
2591996	2023-01-30 23:59:56	1.366667	2.042917	1.410863
2591997	2023-01-30 23:59:57	1.300000	2.093637	1.358937
2591998	2023-01-30 23:59:58	1.300000	2.082795	1.586796
2591999	2023-01-30 23:59:59	1.300000	2.291435	1.544314
2592000	2023-01-31 00:00:00	1.266667	2.084922	1.377082

In the above table, we observe that in using 13 robotic arms, setting (level1, level2, level3) as (0, 1, 12) always has higher capacity then the setting of (0, 12, 1) where only 1 robotic arm is running in level 3 setting.

Therefore, for each fix number of robotic arm, we can calculate the combination that fulfil the success criteria with the minimum operation cost, where the annual operation cost, is calculated as the sum of electric consumption cost and deprecation cost follows:

$$D = \left(\sum_{level=1}^3 count_{level} \cdot power_{level} \right) \cdot (\#Hours \text{ in } 1 \text{ year}) \cdot Tariff + \sum_{level=1}^3 count_{level} \cdot dc_{level} \cdot C$$

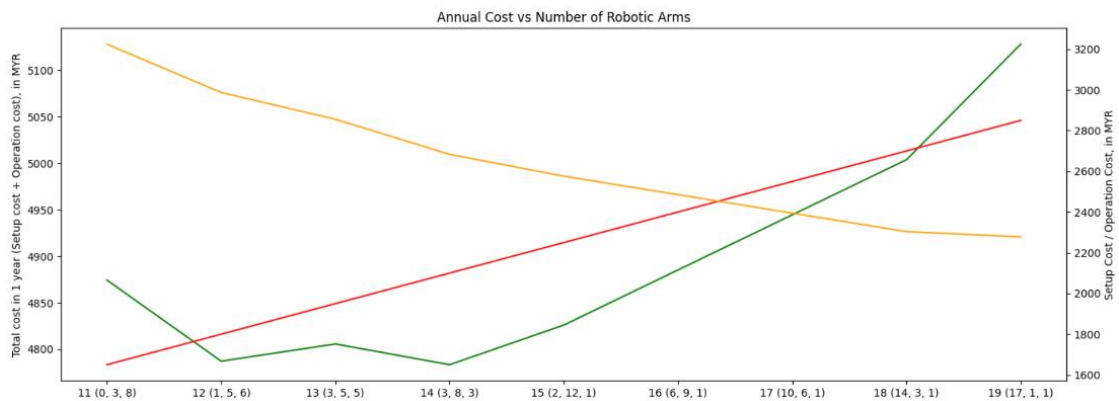
- The power level is 0.02kW, 0.035kW, 0.075kW respectively
- The deprecation coefficient, dc is 0.25, 0.35, 0.6 respectively. This number indicates the loss of equipment if we run the equipment for 1 year at the respective level, the higher the settings that we are running, the higher the deprecation cost.
- Tariff rate used is fix at RM0.38

The total annual cost will then be

$$Total \text{ Annual Cost} = N \cdot C + D$$

Where N is the number of robotic arms, C is the fix setup cost and D is the annual operation cost.

The minimum total annual cost for each number of robotic arms is plotted in the chart below for the case of C = MYR150.



- The optimal number of robotic arms is about 12 to 14 in this case.
- Robotic arms lesser than 11 is not able to support the input rate 99.99% of the time in any combination.
- Setup cost is increased linearly when we increase the number of robotic arms
- Operation cost is lower when we have more robotic arms to adjust for the best efficiency.

16.2 Knapsack Algorithm

The Knapsack algorithm is suggested for optimizing the operation of the robotic arm. The main purpose of the Knapsack in current application is to optimize the material from multiple sources based on the capacity of the robotic arm. There are few assumptions made in this discussion. Firstly, the environment setup is shown below in the Figure 16-1.

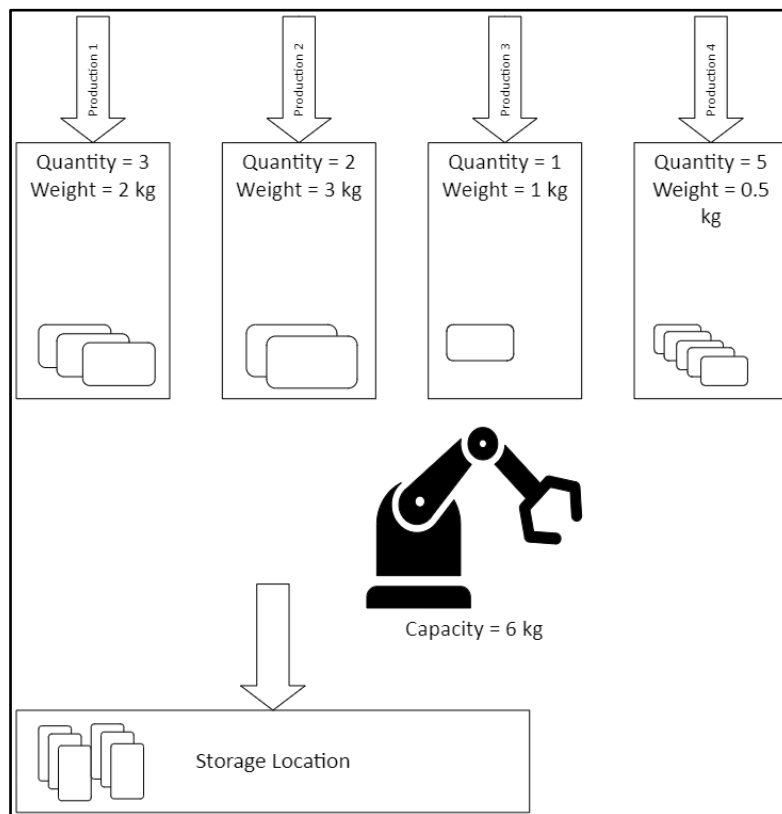


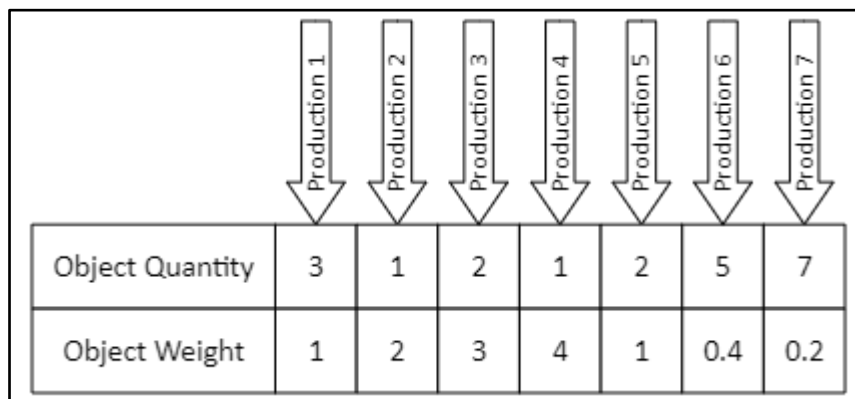
Figure 16-1: Knapsack - Environment Setup.

Given the environment, there is a few conditions that should be noted. The first is the different weight and quantity of rice packages located at each production line. The production 1 is packaging rice package of weight 2 kg and currently there are 3 packages readied for transport. The production 2 has 2 packages of rice with weight 3 kg and so on.

Furthermore, it is assumed that the robotic arm has a capacity of 6 kg which indicates it could carry more than 1 rice package. The consideration of mixed storage for all the different weight of rice packages at the storage location will also be removed as the current objective of the algorithm is to optimize the operation of material transfer based on the robotic arm capacity.

The test for the proposed algorithm in the current context is implemented using the Python Notebook script. There are 2 functions and 1 class object written for the situation. (refer to Appendix for the code) The functions are `knapsack()` and `update_quantity()`. The `knapsack()` is modified (GeeksforGeeks, 2012) so that it will return total weight of the objects that will be picked up by the robotic arm and the object index for the location of the object. In addition, the `update_quantity()` will update the information for the quantity remained in the production lines after the transfer is completed. The class object defined, `manufactureLine`, is used to hold the information of the rice packages in the production line.

To start off with the test, an instance of the object class `manufactureLine` is defined to hold a snapshot of the current rice packages on the production lines. To further accentuate the idea of Knapsack, 7 production lines are defined as shown in Figure 16-2.



	Production 1	Production 2	Production 3	Production 4	Production 5	Production 6	Production 7
Object Quantity	3	1	2	1	2	5	7
Object Weight	1	2	3	4	1	0.4	0.2

Figure 16-2: A snapshot of the production.

The 7 production lines produce different size of rice packages ranging from 0.2 kg to 4 kg. At the point in time that the snapshot is taken, there are different quantity of packages in queue at the end of the production line waiting for transfer.

Then, the function `knapsack()` is run with the quantity and weight of rice packages as parameters and the total weight, $weight \times quantity$, and the object index or production line number will be return to indicate the amount and location of the rice packages that will be picked up by the robotic arm. Lastly, the new quantity remaining on the production will be updated to indicate new batch of rice packages that require transfer. The flow of the operation is shown in Figure 16-3.

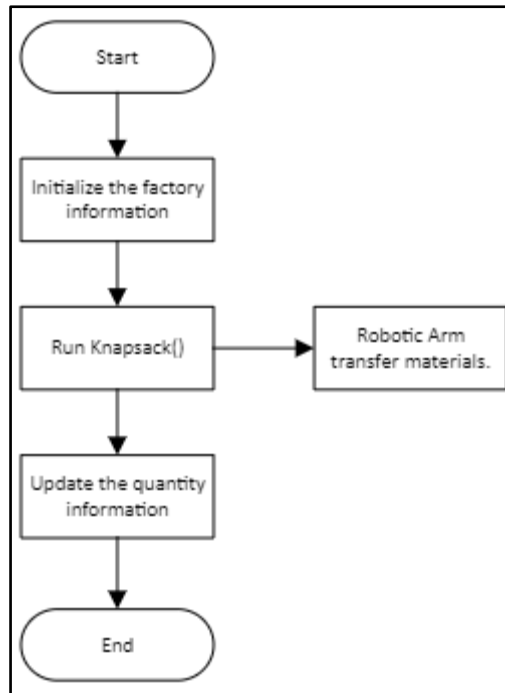


Figure 16-3: Knapsack Implementation for Material Transfer.

The result of the implementation is shown in the Table 6. The first transfer done by the robotic arm is on production line 2, 5, and 6 with a total of 8 packages being picked up. Subsequent runs ensure a maximum usage of the capacity 6 kg for the robotic.

Table 6: Knapsack Operation Result.

Transfer	Total Weight (kg)	Number of Rice Packages	Production Line
1	6	8	[2, 5, 6]
2	6	2	[3]
3	5.4	8	[4,7]
4	3	3	[1]

Thus, it is shown that the Knapsack can be used to optimize the operation where each round trip of the robotic arm will fully utilize the capacity of the robotic arm and cut down on operating cost.

16.3 Travelling Salesman Problem

The traveling salesman is a famous problem in path optimization. The algorithm aims to find the shortest possible distance for the salesman to traverse to all cities. Inspired by this, the solution is implemented in this robotic arm use case. Since the system is able to identify coordinates of all the materials needed to be picked up, the idea is to get the most optimized path for the robotic arm to navigate to each item based on the coordinates being processed. Multiple algorithms can be used to solve this such as the Greedy and Genetic algorithms. As a proof-of-concept, the Brute-Force algorithm is chosen for the simplest implementation.

The algorithm has a main function called `best_path_order`. It takes a list of coordinates as an input and returns the order of the coordinates that results in the shortest total distance travelled by visiting each coordinate once and returning to the starting point. By default, the second input parameter for this function is set to `True` which identifies the first coordinates as the starting point of the robotic arm unless being defined otherwise.

The first step of the function is to define a helper function called "distance" that calculates the Euclidean distance between two points. It takes two coordinates (x1, y1) and (x2, y2) as input and returns the square root of the sum of the squares of the differences between the x and y points.

After that, the function generates all possible permutations of the order of the coordinates using the `permutations()` function from the `itertools` Python module. For each permutation, it calculates the total distance travelled by adding up the distance between each consecutive pair of coordinates in the permutation. It keeps track of the shortest distance found so far and the corresponding permutation.

```

First coordinate is the starting point
Checking: ((1, 2), (5, 6), (3, 4), (0, 1)) . Total distance: 12.727922061357855
Checking: ((1, 2), (5, 6), (0, 1), (3, 4)) . Total distance: 16.97056274847714
Checking: ((1, 2), (3, 4), (5, 6), (0, 1)) . Total distance: 12.727922061357855
Checking: ((1, 2), (3, 4), (0, 1), (5, 6)) . Total distance: 14.142135623730951
Checking: ((1, 2), (0, 1), (5, 6), (3, 4)) . Total distance: 11.313708498984761
Checking: ((1, 2), (0, 1), (3, 4), (5, 6)) . Total distance: 8.48528137423857
('Best path: ', ((1, 2), (0, 1), (3, 4), (5, 6)), '. Total distance: ', 8.48528137423857)

First coordinate is not the starting point
Checking: ((1, 2), (5, 6), (3, 4), (0, 1)) . Total distance: 12.727922061357855
Checking: ((1, 2), (5, 6), (0, 1), (3, 4)) . Total distance: 16.97056274847714
Checking: ((1, 2), (3, 4), (5, 6), (0, 1)) . Total distance: 12.727922061357855
Checking: ((1, 2), (3, 4), (0, 1), (5, 6)) . Total distance: 14.142135623730951
Checking: ((1, 2), (0, 1), (5, 6), (3, 4)) . Total distance: 11.313708498984761
Checking: ((1, 2), (0, 1), (3, 4), (5, 6)) . Total distance: 8.48528137423857
Checking: ((5, 6), (1, 2), (3, 4), (0, 1)) . Total distance: 12.727922061357855
Checking: ((5, 6), (1, 2), (0, 1), (3, 4)) . Total distance: 11.313708498984761
Checking: ((5, 6), (3, 4), (1, 2), (0, 1)) . Total distance: 7.0710678118654755
Checking: ((5, 6), (3, 4), (0, 1), (1, 2)) . Total distance: 8.485281374238571
Checking: ((5, 6), (0, 1), (1, 2), (3, 4)) . Total distance: 11.313708498984761
Checking: ((5, 6), (0, 1), (3, 4), (1, 2)) . Total distance: 14.142135623730951
Checking: ((3, 4), (1, 2), (5, 6), (0, 1)) . Total distance: 15.556349186104047
Checking: ((3, 4), (1, 2), (0, 1), (5, 6)) . Total distance: 11.313708498984761
Checking: ((3, 4), (5, 6), (1, 2), (0, 1)) . Total distance: 9.899494936611667
Checking: ((3, 4), (5, 6), (0, 1), (1, 2)) . Total distance: 11.313708498984761
Checking: ((3, 4), (0, 1), (1, 2), (5, 6)) . Total distance: 11.313708498984761
Checking: ((3, 4), (0, 1), (5, 6), (1, 2)) . Total distance: 16.97056274847714
Checking: ((0, 1), (1, 2), (5, 6), (3, 4)) . Total distance: 9.899494936611665
Checking: ((0, 1), (1, 2), (3, 4), (5, 6)) . Total distance: 7.0710678118654755
Checking: ((0, 1), (5, 6), (1, 2), (3, 4)) . Total distance: 15.556349186104045
Checking: ((0, 1), (5, 6), (3, 4), (1, 2)) . Total distance: 12.727922061357855
Checking: ((0, 1), (3, 4), (1, 2), (5, 6)) . Total distance: 12.727922061357855
Checking: ((0, 1), (3, 4), (5, 6), (1, 2)) . Total distance: 12.727922061357855
('Best path: ', ((5, 6), (3, 4), (1, 2), (0, 1)), '. Total distance: ', 7.0710678118654755)

```

At the end, the function returns the permutation with the shortest distance, which represents the order of the coordinates that results in the optimal path to be used.

17 Summary

In summary, the project achieved the three main objectives. The robotic arm is built for the hardware with the servo motors and the 6DOF aluminium robotic arm frame. In addition, the Firmata protocol is used to relay the motion instruction from host computer to Arduino UNO R3 board, which then converts the instruction to PWM signals to the servo motors. The calculation and optimization are performed on the host computer which has greater computing and memory resources and allow for the use of Python programming language.

Furthermore, the GUI is developed with the use of Windows Forms Application in .NET environment. The use of computer vision, OpenCV, is also implemented for the coordinate generation through the camera and pattern recognition.

The optimization algorithm is also implemented to optimize the operation of the system through the optimization of path and object transfer with the use of Knapsack algorithm and Traveling Salesman algorithm. The Knapsack algorithm optimizes the material transfer based on the weight capacity of the robotic arm.

The contribution from this study is that a foundational code and program had been developed for the system and further research can be conducting in bridging the gap within the study. These gap in the study are the integration between the program between GUI and Arduino. Furthermore, the integration of the optimization algorithm for the robotic control will need further development.

18 Authors Contribution

No	Section	Authors
1	Introduction	Chong Chia Hsing, Hafiidz
2	Project Overview	Chong Chia Hsing
3	User Persona and Characteristics	Hafiidz
4	Hardware Requirements	Hafiidz, Chong Chia Hsing
5	Software Requirements	Maisarah, Chong Chia Hsing
6	System Architecture	Chong Chia Hsing
7	External Interface Requirements	Lei Yulin, Hafiidz, Chong Chia Hsing
8	Use Cases	Maisarah
9	Functional Requirements	Chong Chia Hsing
10	Robotic Arm	Lei Yulin, Hafiidz, Chong Chia Hsing, Gary
11	Optimization	Gary, Hafiidz, Lee Wai Key, Chong Chia Hsing, Maisarah
12	Summary	Chong Chia Hsing, Hafiidz

19 References

GeeksforGeeks. (19 March, 2012). *0-1 Knapsack Problem | DP-10 - GeeksforGeeks*. Retrieved from GeeksforGeeks: href="https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/"

Rice Processing. (24 August, 2020). *Analysis on Rice Industry in Malaysia: Supply, Demand, Challenges and Opportunity*. Retrieved from Rice Milling: <https://rice-processing.com/analysis-on-rice-industry-in-malaysia.html>

20 Appendix

```
def update_quantity(object_index, object_quantity):  
    for i in object_index:  
        object_quantity[i-1] = 0  
    return object_quantity
```

```
class manufactureLine:  
    def __init__(self, quantity, weight):  
        self.quantity = quantity  
        self.weight = weight
```

```

def knapSack(capacity, weight, quantity, n):
    """ A naive recursive implementation of 0-1 Knapsack Problem
    Parameters
    -----
    capacity : int
        Capacity of Knapsack
    weight : list<int>
        Weight of each object
    quantity : list<int>
        The quantity of object
    n : int
        Number of object type / length of the quantity or weight list
    Returns
    -----
    totalWeight: int
        Maximum value of that can be put in a knapsack of capacity W
    objectIndex: list<int>
        A list of integer for the index of object that are chosen.
    """

    # Base Case
    if n == 0 or capacity == 0:
        return 0, [-999]

    # If weight of the nth item is
    # more than Knapsack of capacity W,
    # then this item cannot be included
    # in the optimal solution
    totalWeight_init = weight[n-1]*quantity[n-1]
    objectIndex_init = [n]
    if (totalWeight_init > capacity):
        return knapSack(capacity, weight, quantity, n-1)

    # return the maximum of two cases:
    # (1) nth item included
    # (2) not included
    else:
        totalWeight_case1, objectIndex_case1 = knapSack(capacity-
totalWeight_init, weight, quantity, n-1)
        totalWeight_case2, objectIndex_case2 = knapSack(capacity, weight,
quantity, n-1)
        case1_weightCombined = totalWeight_init + totalWeight_case1
        case1_objectIndexCombined = objectIndex_init + objectIndex_case1
        if case1_weightCombined >= totalWeight_case2:
            return case1_weightCombined, case1_objectIndexCombined
        else:
            return totalWeight_case2, objectIndex_case2

```