Industry 4.0: Robotic Arm Implementation

Sri Akhilesh Pampana
Pavel Chatterjee
Chakli Nagaraja Rahul

Table of contents

Abstract

The robotic arm is used as a kind of symbolic process which adds information to products on a pallet (for instance labeling or tagging products). The Robot should communicate with the conveyor and perform pick and place operations based on the communicated data.

Introduction

The major components of the Industry 4.0 laboratory are listed below:-

1. Conveyor System
2. Turntables
3. Automatic High Bay Warehouse
4. Automated Guided Vehicles
5. **Robotic Arm** & more

Robotic arm is a part of the Industry 4.0 system

The primary use of the Braccio Robotic arm is for lifting and placing of goods from conveyor to storage and vice versa. Based on the type of operation required, it either picks items from the conveyor pallet to the empty placeholders in the storage area or it picks from the storage area(Products with specific product id's) and places them on the conveyor pallet.
Another use can be to differentiate the type of products arriving. Let's say there are 3 types of products as follows:
Product A: 3 items
Product B: 2 items
Product C: 3 items
In order to differentiate the products, segregation of the items is possible with the help of the Robotic Arm
The Robotic arm implementation was possible with both

1. Hardware Components:- E.g. Braccio Robotic Arm, Arduino Mega 2560
2. Software Components:- E.g. Arduino IDE v1.8.2

A more detailed insight into the specifications and working of all the components has been covered in [1]
For now the Robotic arm will be used in the logistic laboratory to have a learning environment of Industry 4.0. From the observation, the benefits and drawbacks of Industry 4.0 can be analyzed.
From the analogies derived such devices are to be used in all factories that can use Industry 4.0 (E.g. Food processing, Car manufacturing, etc.)

Component Description

Hardware components:

The components used:

1. Braccio Robotic Arm
2. Arduino Mega 2560.
3. Arduino compatible Shield.

Braccio Robotic Arm:

Italian for arm, the Braccio is a do-it-yourself robotic arm kit which you build from a box of parts and is controlled using the shield with an Arduino board. The arm consists of a total of 6 servo motors:

- M1 – Base
- M2 – Shoulder
- M3 – Elbow
- M4 – Vertical wrist
- M5 – Rotary wrist
- M6 – Grabber

Servomotors and Types.

A servomotor is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration. It consists of a suitable motor coupled to a sensor for position feedback. It also requires a relatively sophisticated controller, often a dedicated module designed specifically for use with servomotors. These are of two different types of servo motors such as SR311 and SR431

SR311

Control Signal	PWM Analog
Torque	@ 4.8V: 43.13 oz-in (3.1 kg-cm) @ 6.0V: 52.86 oz-in (3.8 kg-cm)
Weight	0.95 oz (27.0 g)
Dimensions	1.23×0.65×1.13 in (31.3×16.5×28.6 mm)
Speed	@ 4.8V: 0.14 sec/60° @ 6.0V: 0.12 sec/60°
Rotation Support	Dual Bearings
Gear Material	Metal
Rotation Range	180°
Connector Type	J (aka Futaba)

Figure 1: Servo motor SR311.

SR431

Control Signal	PWM Analog
Torque	@ 4.8V: 169.5 oz-in (12.2 kg-cm) @ 6.0V: 201.4 oz-in (14.5 kg-cm)
Weight	2.19 oz (62.0 g)
Dimensions	1.65×0.81×1.56 in (42.0×20.5×39.5 mm)
Speed	@ 4.8V: 0.20 sec/60° @ 6.0V: 0.18 sec/60°
Rotation Support	Dual Bearings
Gear Material	Metal
Rotation Range	180°
Connector Type	J (aka Futaba)

Figure 2: Servo motor SR431

Arduino Mega 2560.

- Arduino Mega 2560 PIN diagram

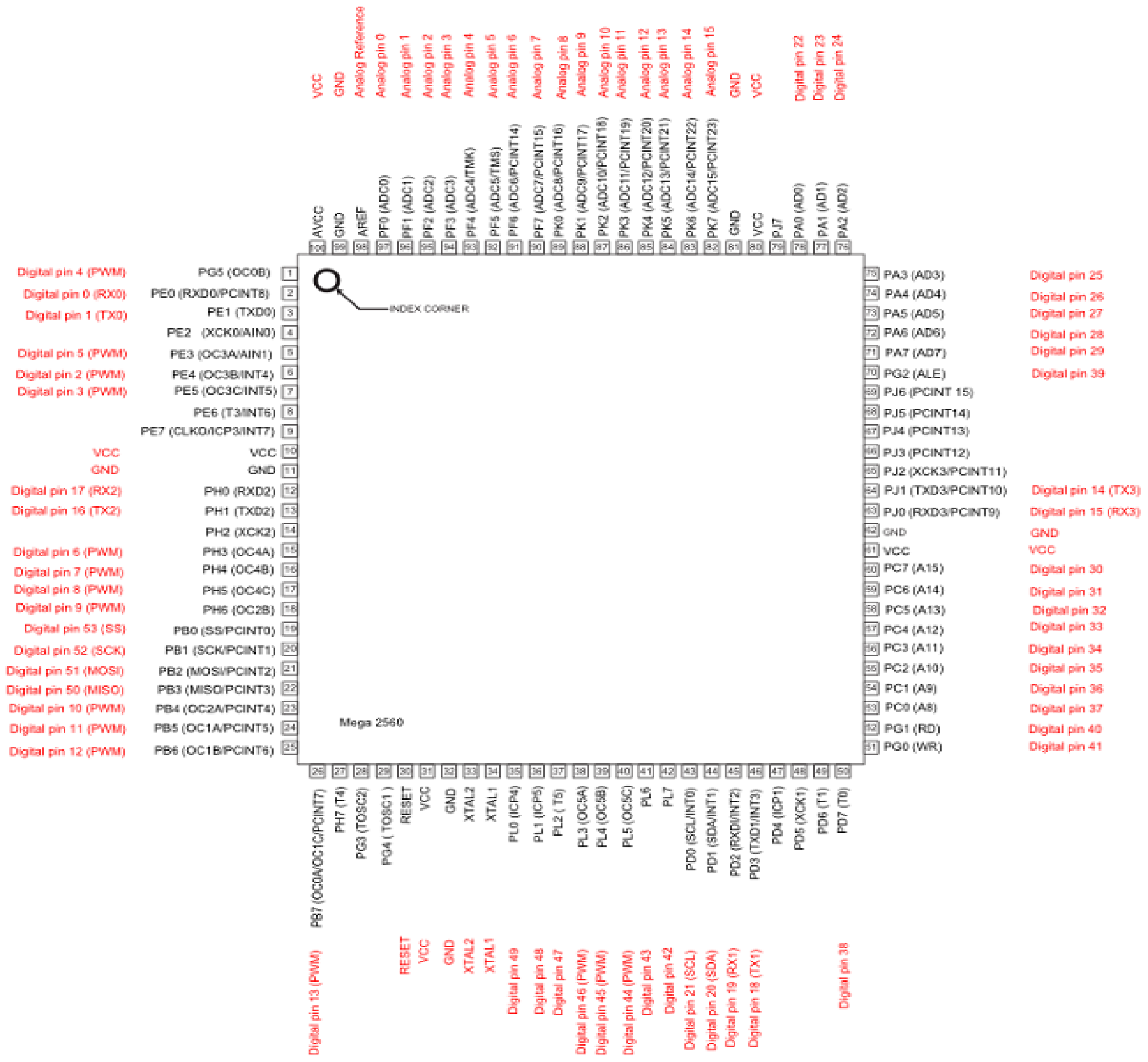


Figure 3: Pin diagram of Atmega 2560

Arduino compatible Shield:

Arduino Compatible Shields. Arduino compatible boards are at minimum, hardware-compatible with certain Arduino shields based upon the header, or pin-out connector. Most newer “Arduino compatible” boards claim to be compatible with the same shields that the Arduino

SOFTWARE:

The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the GND and Vin pin headers of the POWER connector. The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may become unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.
The power pins are as follows:

- Vin.: The input voltage to the board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- 5V: this pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- 3V3: A 3.3-volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- GND: Ground pins.
- IOREF: This pin on the board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.

Arduino Software (IDE)

- There are two main functions in the Braccio library, these are Braccio.servoMovement and Braccio.begin. Braccio.begin initializes and sets up the initial position for Braccio Arm. All the servo motors are positioned in the "safety" position with each angle as follows:

M1: Base	– 90 degrees
M2: Shoulder	– 45 degrees
M3: Elbow	– 180 degrees
M4: Wrist vertical	– 180 degrees
M5: Wrist rotation	– 90 degrees
M6: Gripper	– 10 degrees

- Braccio.servoMovement allows you to control all the Braccio's servos with only one command, including a millisecond delay between the movement of each servo at the beginning of each step and then the angle at which each motor should move. The parameters for the step delay and each motor can be found below:

Step Delay: Values from 10 to 30 milliseconds.

M1: Base degrees	-	Range between 0 to 180 degrees
M2: Shoulder degrees	-	Range between 15 to 165 degrees
M3: Elbow degrees	-	Range between 0 to 180 degrees
M4: Wrist vertical degrees	-	Range between 0 to 180 degrees
M5: Wrist rotation degrees	-	Range between 0 to 180 degrees
M6: Gripper degrees	-	Range between 10 to 73 degrees.

(A value of 10 means the gripper is open and a value of 73 means the gripper is closed).

Working:

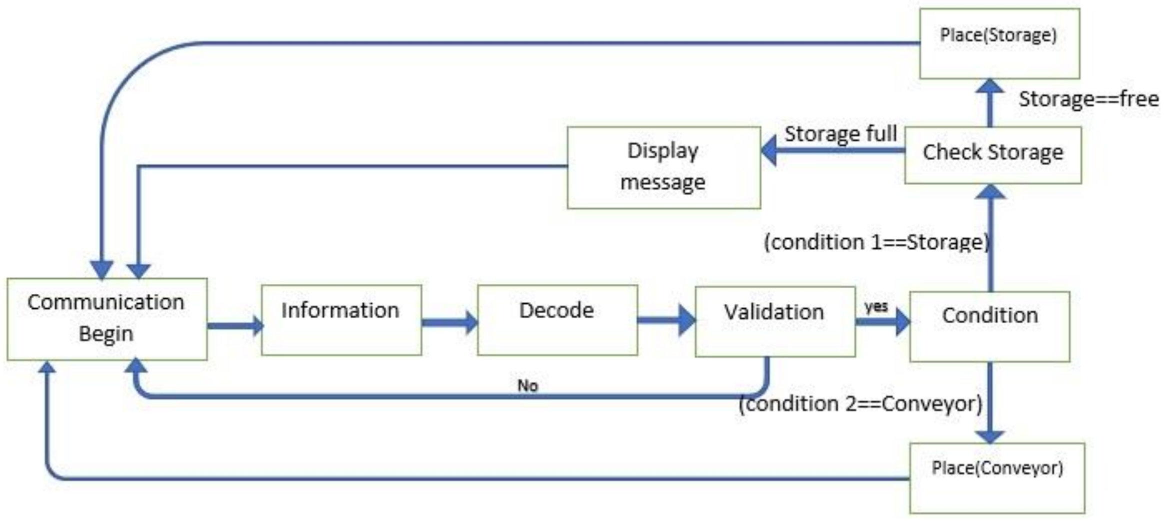


Figure 4: Basic Block Diagram

Information arrives in the form of bytes with every pallet arrival. The bytes have information about the orientation of the pallet (North, South, East and West), position, the type of operation, a unique product ID and the checksum value. In decoding, the bytes are added and compared in the validation block. In case the validation block fails, the control goes back to the beginning of communication and waits for the next set of data from the pallet arrival. In case the validation step is true, based on the condition the data set has it performs the following:

1. Pick up from conveyor and placing in Storage
   It checks if the storage is full or not. Two conditions arise, i.e. if the storage is full, it prompts a message "Storage is full" and goes back to beginning of communication. If the storage is not full it places the product in the storage
2. Placing a product from storage to conveyor
   It checks if the product is already there in the storage and then picks the product from the storage and places it in the pallet on the conveyor in a position that's mentioned in the byte information.

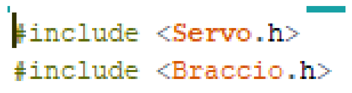
Technical Aspects

Main Code Programming

The basic code can be explained as the following way:
The protocol for the code for accepting data from the conveyor is as follows:
Conveyor (sends 5 Bytes when a pallet arrived):

1. Byte: Orientation (0=N, 1=W, 2=S, 3=E)
2. Byte: Position (1/2/3)
3. Byte: Type of operation to be performed (1=pick from the pallet, put it somewhere into the storage / 2 pick it from the storage -> put it to the specified pallet position)
4. Byte: Product ID (globally unique)
5. Byte: Checksum (sum of the first 4 bytes)

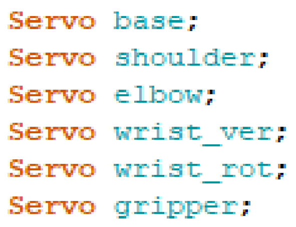
Example:
(0 / 2 / 0 / 147 / 149) -> take product 147 from position 2 of the north-headed pallet and put it somewhere into the storage.
Code begins with the HEADER files #include<Servo.h> and #include<Braccio.h>. The library of #include<Braccio.h> can be referred from braccio website and can be added to the code from Sketch => include library.



```
#include <Servo.h>
#include <Braccio.h>
```

Figure 5

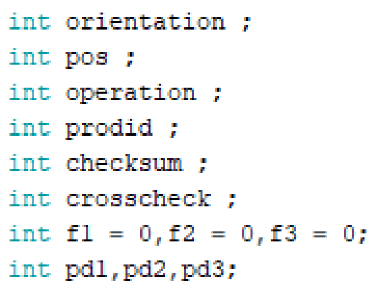
Once the library of Braccio is included in the code, the variables for the motors of the robot are automatically included.



```
Servo base;
Servo shoulder;
Servo elbow;
Servo wrist_ver;
Servo wrist_rot;
Servo gripper;
```

Figure 6

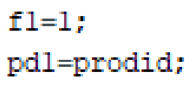
Then the declaration of integer type of variables for orientation of the pallet(orientation), position of object in the pallet(pos), type of operation(operation), product ID(prodid), checksum(checksum), crosscheck for checking the received data is correct or not(crosscheck), and positions on storage (f1, f2, f3 for storage 1st position, 2nd position and 3rd position respectively. These variables are declared initially to 0.)



```
int orientation ;
int pos ;
int operation ;
int prodid ;
int checksum ;
int crosscheck ;
int f1 = 0,f2 = 0,f3 = 0;
int pd1,pd2,pd3;
```

Figure 7

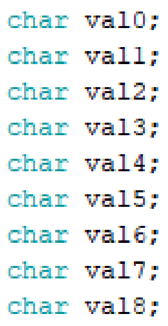
Once the robot places any object in any of the storage position the variables are forced to store 1 which means that f1 = 0 indicates 1st position in storage is free and f1 = 1 indicates 1st position in storage is filled).
Product ID's stored in storage area are declared (pd1, pd2, pd3 for product ID in 1st position in storage, product ID in 2nd position in storage and product ID in 3rd position in storage respectively).



```
f1=1;
pd1=prodid;
```

Figure 8

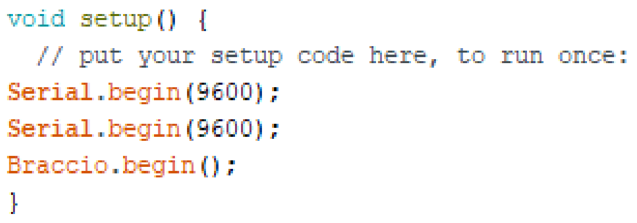
The declaration of character type of variables as val0 (for storage of byte for orientation), val1 (for storage of byte for position), val2 (for storage of byte for operation), val3, val4, val5 (val3, val4 and val5 for storage of byte for product ID), val6, val7, and val8 (for storage of byte for checksum).



```
char val0;
char val1;
char val2;
char val3;
char val4;
char val5;
char val6;
char val7;
char val8;
```

Figure 9

The code be initiated with Serial.begin(9600) and Braccio.begin() to initialize serial communication and libraries for the robotic arm to function with Arduino.

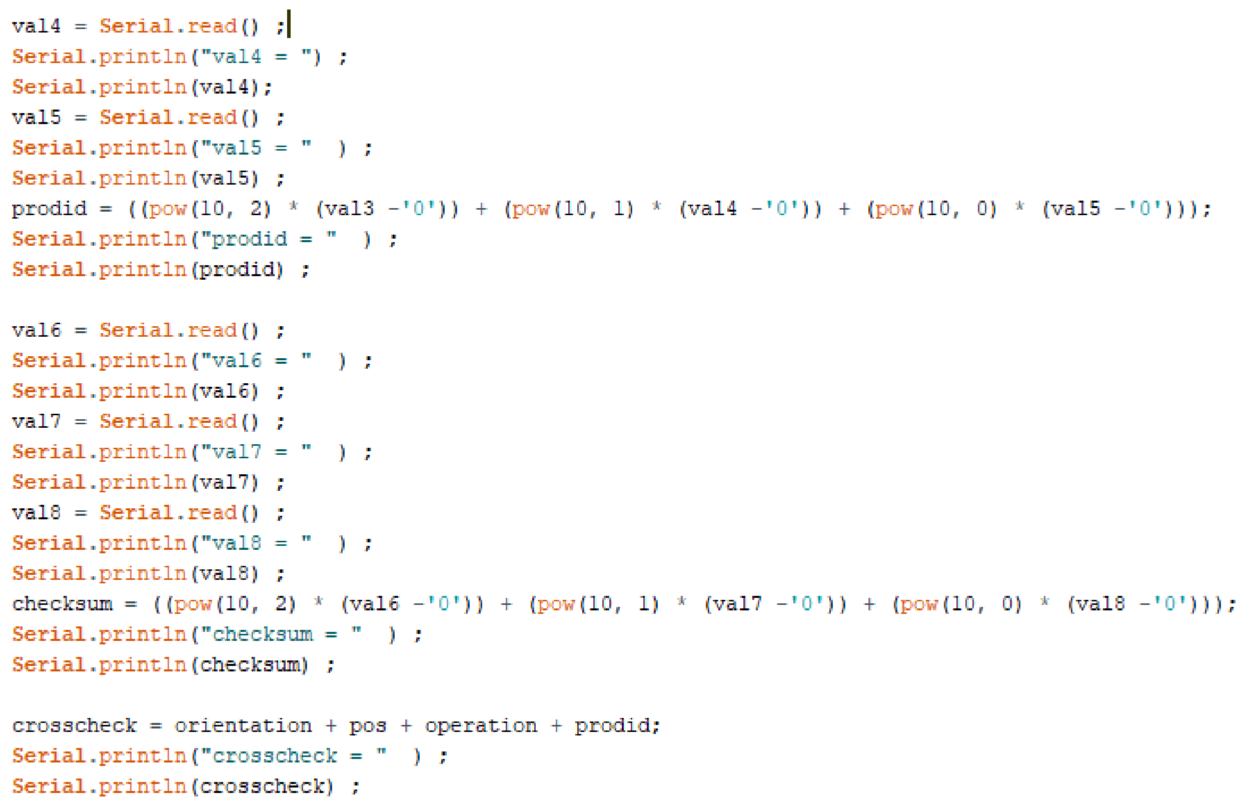


```
void setup() {
  // put your setup code here, to run once:
Serial.begin(9600);
Serial.begin(9600);
Braccio.begin();
}
```

Figure 10

Arduino waits till the data from the conveyor is received and then stores the information in variables.
The char values are converted to integer type and stored in respective integer type variables. Variable crosscheck is used to check whether received data is correct or not.

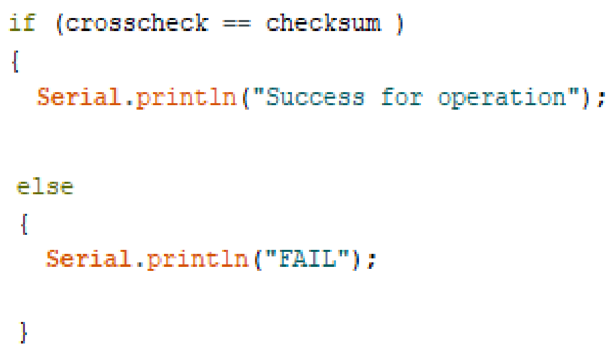
```
 if(Serial.available() == 9)
 {

val0 = Serial.read() ;
Serial.println("val0 = "  ) ;
Serial.println(val0) ;
orientation = val0 - '0';
Serial.println("orientation = "  ) ;
Serial.println(orientation) ;

val1 = Serial.read() ;
Serial.println("val1 = ") ;
Serial.println(val1);
pos = val1 - '0';
Serial.println("pos = "  ) ;
Serial.println(pos) ;

val2 = Serial.read() ;
Serial.println("val2 = ")  ;
Serial.println(val2);
operation = val2 - '0';
Serial.println("operation = "  ) ;
Serial.println(operation) ;

val3 = Serial.read() ;
Serial.println("val3 = ") ;
Serial.println(val3);
```



```
val4 = Serial.read() ;
Serial.println("val4 = ") ;
Serial.println(val4);
val5 = Serial.read() ;
Serial.println("val5 =  "  ) ;
Serial.println(val5) ;
prodid = ((pow(10, 2) * (val3 -'0')) + (pow(10, 1) * (val4 -'0')) + (pow(10, 0) * (val5 -'0')));
Serial.println("prodid = "  ) ;
Serial.println(prodid) ;

val6 = Serial.read() ;
Serial.println("val6 = "  ) ;
Serial.println(val6) ;
val7 = Serial.read() ;
Serial.println("val7 = "  ) ;
Serial.println(val7) ;
val8 = Serial.read() ;
Serial.println("val8 = "  ) ;
Serial.println(val8) ;
checksum = ((pow(10, 2) * (val6 -'0')) + (pow(10, 1) * (val7 -'0')) + (pow(10, 0) * (val8 -'0')));
Serial.println("checksum = "  ) ;
Serial.println(checksum) ;

crosscheck = orientation + pos + operation + prodid;
Serial.println("crosscheck = "  ) ;
Serial.println(crosscheck) ;
```

Figure 11

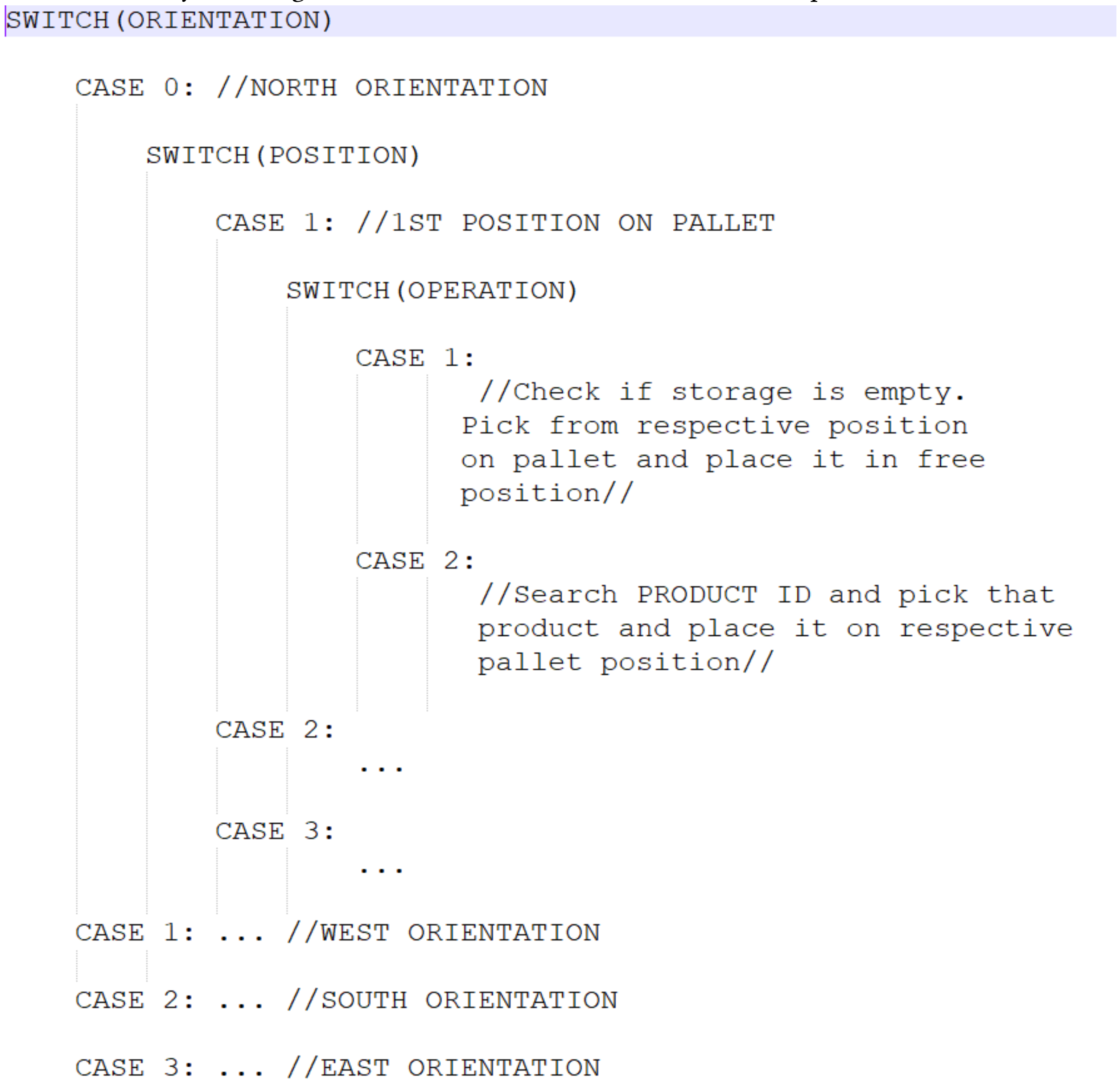
After storing, Arduiuno checks whether the data received is correct or not using the variable crosscheck as per the previous image. If the data is correct the execution process starts or the Arduino sends failed message.



```
if (crosscheck == checksum )
{
  Serial.println("Success for operation");


 else
 {
   Serial.println("FAIL");


 }
```

Figure 12

The main part of the code uses SWITCH cases as the data must be compared and executed as per the data received from the conveyor. Code has nested Switch for orientation, position and operation. Switch case for orientation has four cases and each Switch case for orientation has three cases of position and each Switch case for position has two switch cases of operation. In this nested Switch cases arduino will go to a particular case depending on the data received from the conveyor and gives the command to the robot to do the required task.



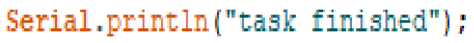
```
SWITCH(ORIENTATION)

    CASE 0: //NORTH ORIENTATION

        SWITCH(POSITION)

            CASE 1: //1ST POSITION ON PALLET

                SWITCH(OPERATION)

                    CASE 1:
                           //Check if storage is empty.
                          Pick from respective position
                          on pallet and place it in free
                          position//

                    CASE 2:
                           //Search PRODUCT ID and pick that
                           product and place it on respective
                           pallet position//

            CASE 2:
                    ...

            CASE 3:
                    ...

    CASE 1: ... //WEST ORIENTATION

    CASE 2: ... //SOUTH ORIENTATION

    CASE 3: ... //EAST ORIENTATION
```

Figure 13

For example: The data received from the conveyor is as follows: 011111113.

This indicates that the pallet is in north position and the robot must pick from 1st position of the pallet and store it in the free place in storage. The data can be identified as orientation = 0, pos = 1; operation = 1; prodid = 111 and checksum = 113.

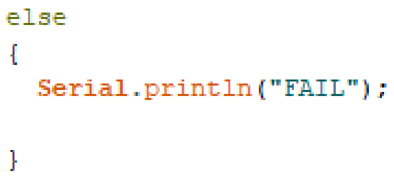
Another example can be explained as follows: The data received from the conveyor is as follows: 312111117. This indicates that the robot must pick object from storage with product ID 111 and place it in the 1st position in the pallet (WEST orientation) on conveyor. The data can be identified as orientation = 3, pos = 1; operation = 2; prodid = 111 and checksum = 117. Once the task is done by the robot, Arduino will send an acknowledgement to the conveyor that is the task is completed.



```
Serial.println("task finished");
```

Figure 14

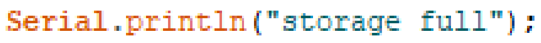
The other acknowledgment is that data received is corrupt and the checksum is not matching with crosscheck. In this case Arduino will ask conveyor to send data once again.



```
else
{
  Serial.println("FAIL");

}
```

Figure 15

If the storage is full then arduino will send an acknowledgement that the storage is full and it cannot perform and further operation.

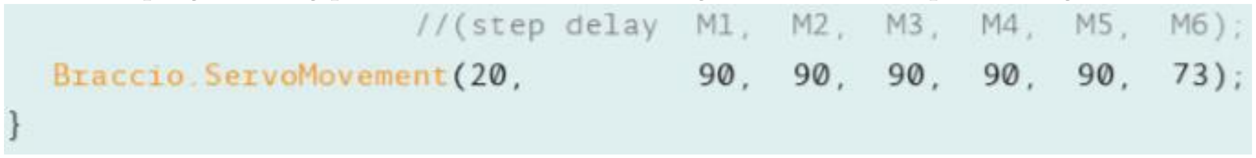


```
Serial.println("storage full");
```

Figure 16

Robotic Movements

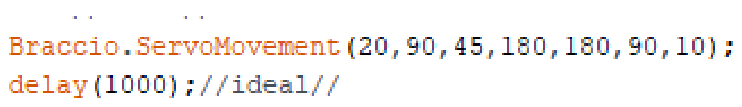
Robotic programming part can be understood using the basic motor positioning.



```
                        //(step delay  M1,  M2,  M3,  M4,  M5,  M6);
  Braccio.ServoMovement(20,            90,  90,  90,  90,  90,  73);
}
```

Figure 17

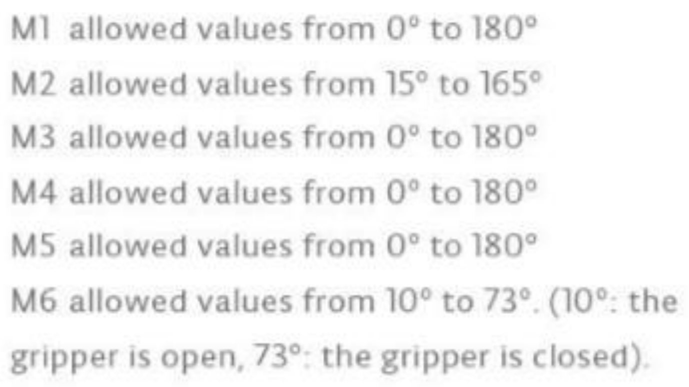
M1, M2, M3, M4, M5 and M6 refers to base, shoulder, elbow, wrist_ver, wrist_rot and gripper respectively. The step delay can be in milliseconds delay between the movement of each motor and the allowed values are from 10 to 30 msec.
The robot's motors position for the ideal position can be given as:



```
Braccio.ServoMovement(20,90,45,180,180,90,10);
delay(1000);//ideal//
```

Figure 18

The maximum and minimum angles for each motor are as follows:



M1 allowed values from 0° to 180°

M2 allowed values from 15° to 165°

M3 allowed values from 0° to 180°

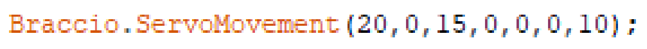
M4 allowed values from 0° to 180°

M5 allowed values from 0° to 180°

M6 allowed values from 10° to 73°. (10°: the gripper is open, 73°: the gripper is closed).

Figure 19

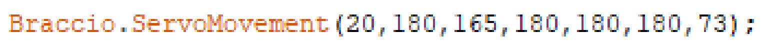
The minimum angles of the motors can be coded as



```
Braccio.ServoMovement(20,0,15,0,0,0,10);
```

Figure 20

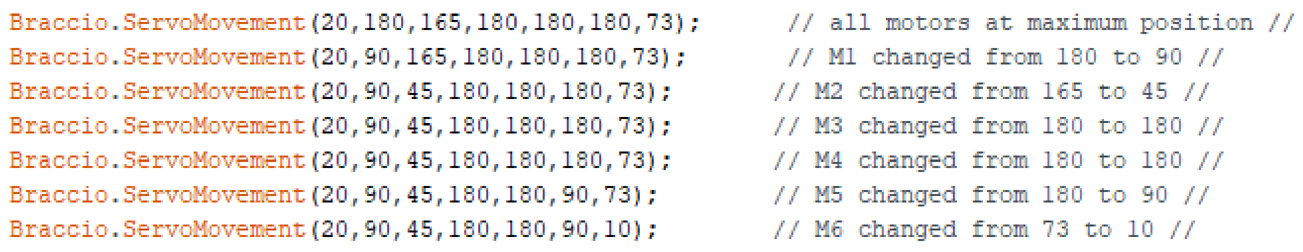
And the maximum angles of the motors can be coded as



```
Braccio.ServoMovement(20,180,165,180,180,180,73);
```

Figure 21

The angles for the motors can be varied between the maximum and minimum angles till the desired position is acquired. For instance to move the motors of the robot from maximum to ideal position, the following steps can be followed:



```
Braccio.ServoMovement(20,180,165,180,180,180,73);      // all motors at maximum position //
Braccio.ServoMovement(20,90,165,180,180,180,73);       // M1 changed from 180 to 90 //
Braccio.ServoMovement(20,90,45,180,180,180,73);       // M2 changed from 165 to 45 //
Braccio.ServoMovement(20,90,45,180,180,180,73);       // M3 changed from 180 to 180 //
Braccio.ServoMovement(20,90,45,180,180,180,73);       // M4 changed from 180 to 180 //
Braccio.ServoMovement(20,90,45,180,180,90,73);        // M5 changed from 180 to 90 //
Braccio.ServoMovement(20,90,45,180,180,90,10);        // M6 changed from 73 to 10 //
```

Figure 22

Integration and further changes to the code

Main code must be integrated with the robotic movements code to get the desired final code. The Serial.println lines in every switch case must be replaced with respective calibrated robotic movements for both picking and placing operations for required positioning on the pallet and storage respectively. The acknowledgements must be replaced with respective real time data as required. The storing of the serial data from the conveyor can be changed accordingly as per requirement.

Challenges

We have a quite few challenges in the project such as

1. Calibration issues with the motors and hence could not perform pick-up related tasks as expected.
2. The base stability was an issue as the base was not fixed, which lead to the problems in picking and placing positions.
3. The communication was not implemented properly as the devices integration requirements were met.

The probable solution could be as requirement of industrial standard motors that can be tuned exactly.

Conclusion

Items were picked from conveyor and placed in storage and vice versa

Future Scope

The future scope of the laboratory is to have a learning environment to demonstrate industry-4.0-technologies as well as their benefits and drawbacks. People who work with the laboratory should be able to design in a short period of time their very own production and logistics system which meets the requirements of a given task. In this way, it should be also possible to compare different system structures and processes in terms of performance and stability.

References

[1] Arduino Mega 2560: (https://store.arduino.cc/arduino-mega-2560-rev3).

[2] Pin diagram of mega 2560: (https://www.arduino.cc/en/Hacking/PinMapping2560)

[3] Software for coding: https://www.arduino.cc/en/Reference/SoftwareSerial

[4] Research Challenges of Industry 4.0 for Quality Management, Innovations in Enterprise Information Systems Management and Engineering: 4th International Conference, ERP Future 2015 - Research, Munich, Germany, November 16-17, 2015, Revised Papers

[5] Design Principles for Industry 4.0 Scenarios,

https://pdfs.semanticscholar.org/8f60/1826b213012ee349534511e9a5b617278e5e.pdf