4/21/2021

Robotic Arm

Supervised by: Dr. Abdelatif

PowerLink Team

Mahmoud Ahmed Abdelghany	Sec: 4 BN: 8
Mohamed Ragab Fergany	Sec: 3 BN: 30
Abd Elrahman Elsayed Abdo	Sec: 2 BN:30
Mario Magdy Wageh	Sec: 3 BN: 16
Ahmed Elsayed Ragab	Sec: 1 BN: 4

i. Abstraction

In this paper, we will cover the basic steps for implementing the robotic arm project, starting from design, through testing, and finally production.

ii. Introduction

This project is one of the most important automatic control projects, especially in the field of industry, as the automatic arm is located in most factories, which facilitates the control process and increases productivity.

iii. Design Process

In this project there are 3 degrees of freedom, so if we want to move to a point in the vacuum then we must use 3 motors to achieve this point in addition to a fourth motor to control the gripper.

So, we have to go through the mechanical design first, then the electrical design, and then combine them together.

iV. Mechanical Design

Choosing the dimensions and the number of joints in the arm, which is one of the very important things, so the design must be studied well, taking into account the strong momentum affecting the arm in the sleep mode and also the force affecting it in the motion position, so we used a ready-made 3D design.

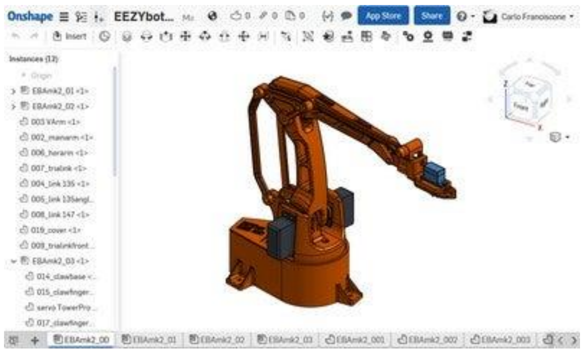


Figure 1: 3d design of robotic arm

This design consists of three main parts:

- The main base
- The main arms
- The gripper

Assembling the main base

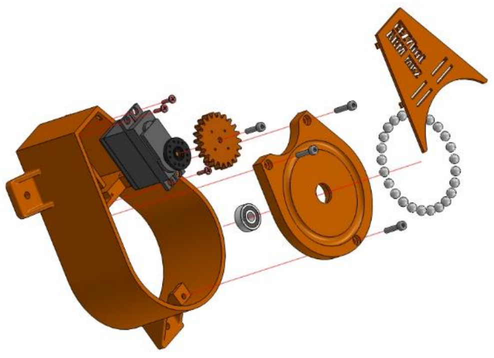


Figure 2: Assembling the main base

Assembling the main arms

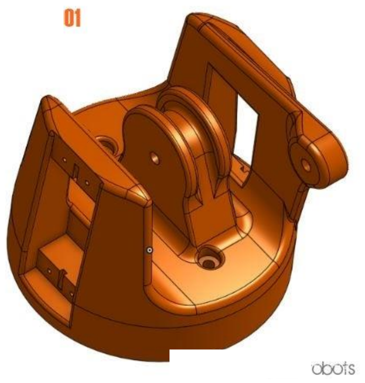


Figure 3: the base part that holding the arm

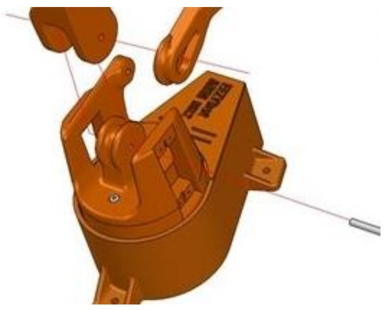


Figure 4: connecting the base and the arms

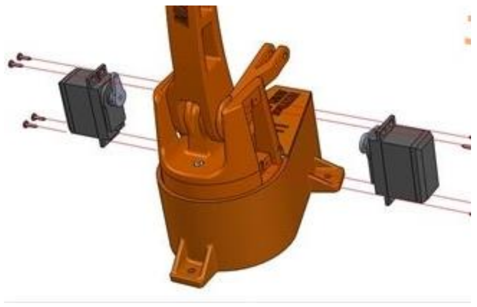


Figure 5: installation of two arm motors

Assembling the gripper

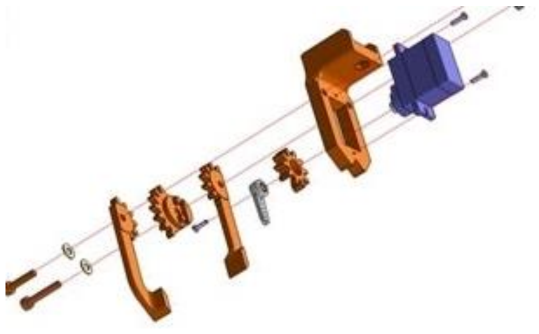


Figure 6: Assembling the gripper

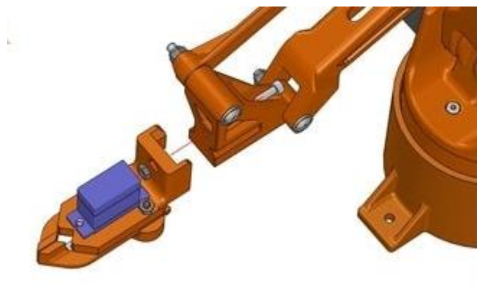


Figure 7:The Gripper

V. Electrical design

Electrical system description

Here comes the electrical design part, which is the soul of the robotic arm. All parts must be carefully selected.

Components:

- 3 Servo motor High Torque 180 degrees (90in each direction) 500mA-900mA.

Figure 8: servo motor high torque

- 1 servo motor low torque 180 degrees (90in each direction).

Figure 9: servo motor low torque

- Arduino nano

This is the main controller it will take the orders from GUI through MODBUS RTU and it will send the electrical signal to the servo motor driver through I^2C.

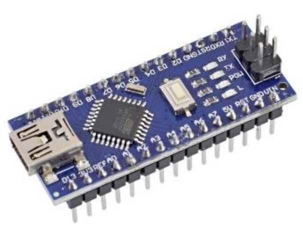


Figure 10: Arduino nano

- Step down converter(3A)
  It will convert the voltage level from 12 volts to 5 volts (the rated voltage of motors).

Figure 11: step down converter

- PCA9586
  16 channel 12-bit PWM servo motor driver module which will take the order from Arduino and generate the PWM signal for the 4 motors, Powered by 5 volts from DC-Dc converter.

Figure 12: 16 channel servo motor driver

Electrical connection

we have two main parts:

- Servo Motor Driver + DC-DC converter (Part1)
- Arduino nano (Part2)

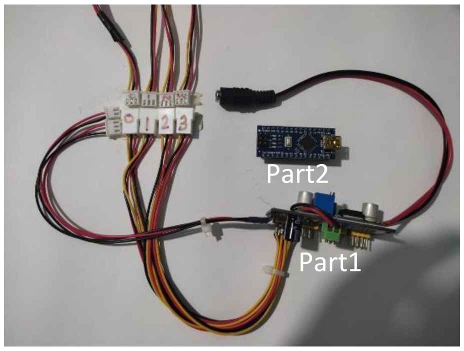


Figure 13: Electrical Wiring

- Part1:

  In this part there is four 3 pin connectors for the motors and they numbered from 0 to 3, and we have two connectors for Arduino one for the power (A) and the other for I^2C communication (B), and the last one is a 12 volts DC power jack.

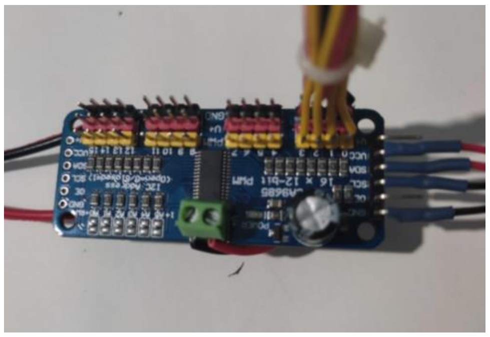


Figure 14: Part1

- Part2

  In this part there is two connectors from part1 one for power and the other for the I^2C communication and we have one (USB-micro USB) to connect the Arduino to the laptop.

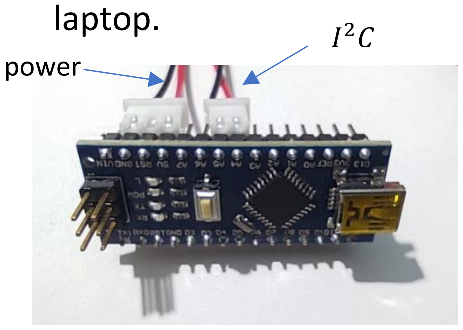
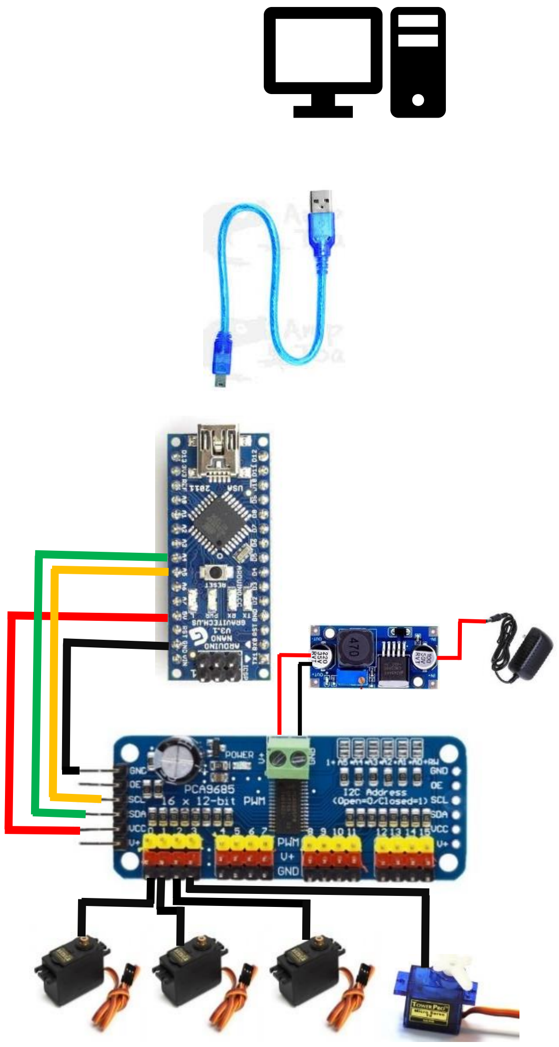


Figure 15: Part2



Vi. Control method

The objective is to achieve a point in a 3d space

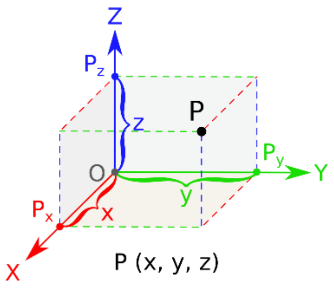


Figure 16: 3D Space

so, we want to control the position of the three main motors:

- Base Motor (90 degrees corresponding to motion in XY plan)
- Right Arm Motor (63 degrees corresponding to motion in ZX plan)
- Left Arm Motor (81 degrees corresponding to motion in YZ direction)

Due to mechanical constrains of the Robotic Arm design the right & left motors are calibrated to their optimum angle values which are 63 degrees and 81 degrees respectively.

PWM Vs Degrees Mapping

These motors are running at PWM signal with duty cycle (2%-12.5%) at frequency 50 Hz so, if we want to achieve a mechanical degree, we can use this relation:

$$\theta_{mech} = \frac{1800}{97}\,\delta_{\%} - \frac{4500}{97}$$

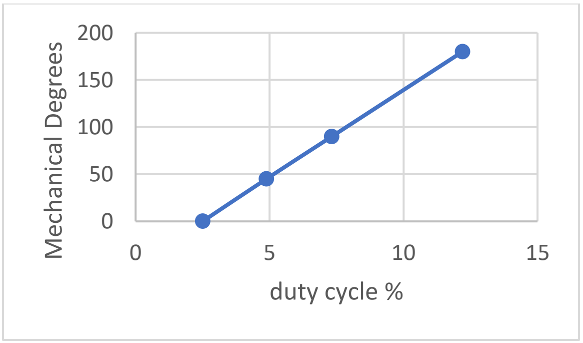


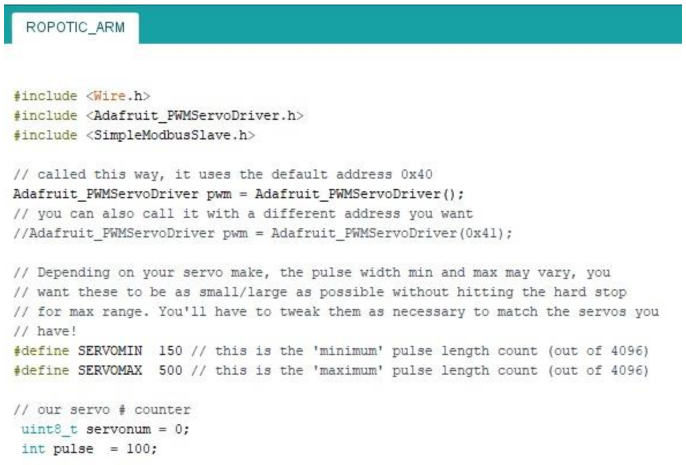
Figure 17: relation between duty cycle and degree

the previous relation is deduced by calibration and tunning.

Vii. Algorithm

We use Arduino nano as our microcontroller to receive the values of the required PWM signals as an input which corresponding to a specific angle from GUI and process our algorithm then send a signal to the servo motor driver as output which the last will supply the required motor with power and control signal.

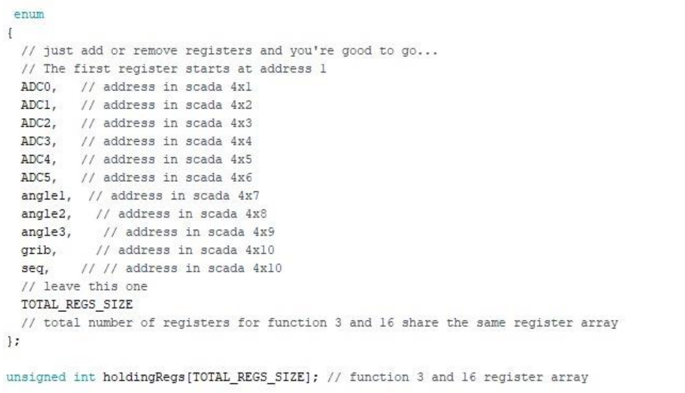
Programming code



ROPOTIC_ARM

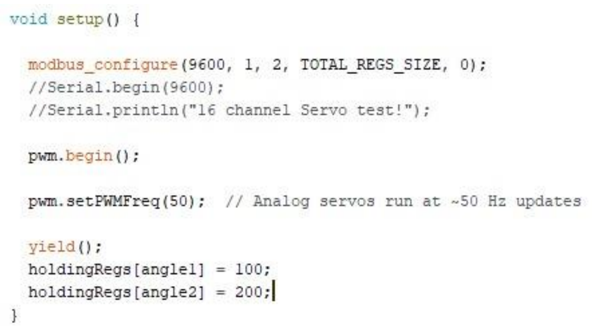
```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
#include <SimpleModbusSlave.h>

// called this way, it uses the default address 0x40
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
// you can also call it with a different address you want
//Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(0x41);

// Depending on your servo make, the pulse width min and max may vary, you
// want these to be as small/large as possible without hitting the hard stop
// for max range. You'll have to tweak them as necessary to match the servos you
// have!
#define SERVOMIN  150 // this is the 'minimum' pulse length count (out of 4096)
#define SERVOMAX  500 // this is the 'maximum' pulse length count (out of 4096)

// our servo # counter
 uint8_t servonum = 0;
 int pulse  = 100;
```

Figure 18: importing libraries & variable declaration



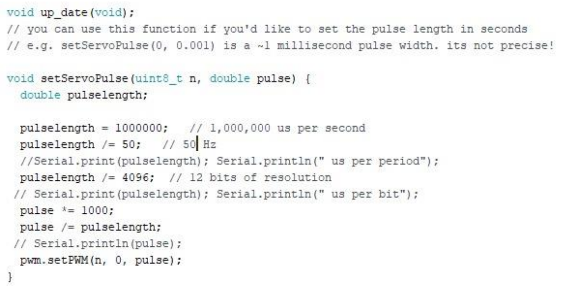
```
 enum
{
  // just add or remove registers and you're good to go...
  // The first register starts at address 1
  ADC0,   // address in scada 4x1
  ADC1,   // address in scada 4x2
  ADC2,   // address in scada 4x3
  ADC3,   // address in scada 4x4
  ADC4,   // address in scada 4x5
  ADC5,   // address in scada 4x6
  angle1,  // address in scada 4x7
  angle2,   // address in scada 4x8
  angle3,    // address in scada 4x9
  grib,     // address in scada 4x10
  seq,    // // address in scada 4x10
  // leave this one
  TOTAL_REGS_SIZE
  // total number of registers for function 3 and 16 share the same register array
};

unsigned int holdingRegs[TOTAL_REGS_SIZE]; // function 3 and 16 register array
```

Figure 19: Holding Registers



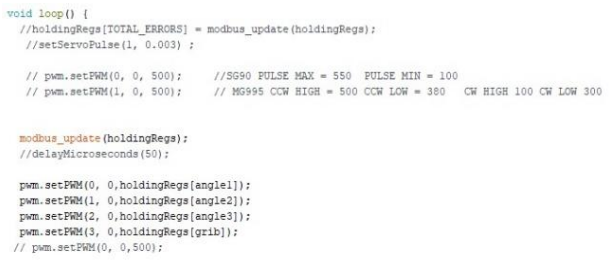
```
void setup() {

  modbus_configure(9600, 1, 2, TOTAL_REGS_SIZE, 0);
  //Serial.begin(9600);
  //Serial.println("16 channel Servo test!");

  pwm.begin();

  pwm.setPWMFreq(50);  // Analog servos run at ~50 Hz updates

  yield();
  holdingRegs[angle1] = 100;
  holdingRegs[angle2] = 200;
}
```

Figure 20: communication protocol initialization



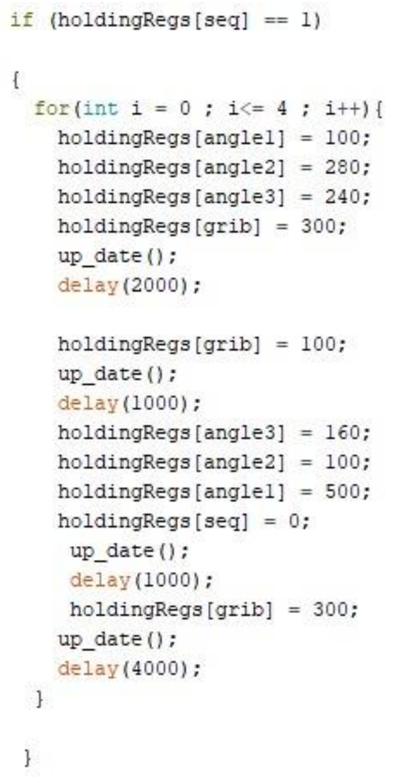
```
void up_date(void);
// you can use this function if you'd like to set the pulse length in seconds
// e.g. setServoPulse(0, 0.001) is a ~1 millisecond pulse width. its not precise!

void setServoPulse(uint8_t n, double pulse) {
  double pulselength;

  pulselength = 1000000;   // 1,000,000 us per second
  pulselength /= 50;   // 50 Hz
  //Serial.print(pulselength); Serial.println(" us per period");
  pulselength /= 4096;  // 12 bits of resolution
 // Serial.print(pulselength); Serial.println(" us per bit");
  pulse *= 1000;
  pulse /= pulselength;
 // Serial.println(pulse);
  pwm.setPWM(n, 0, pulse);
}
```

Figure 21: Holding Registers Update



```
void loop() {
  //holdingRegs[TOTAL_ERRORS] = modbus_update(holdingRegs);
   //setServoPulse(1, 0.003) ;

   // pwm.setPWM(0, 0, 500);     //SG90 PULSE MAX = 550  PULSE MIN = 100
   // pwm.setPWM(1, 0, 500);     // MG995 CCW HIGH = 500 CCW LOW = 380   CW HIGH 100 CW LOW 300


  modbus_update(holdingRegs);
  //delayMicroseconds(50);

  pwm.setPWM(0, 0,holdingRegs[angle1]);
  pwm.setPWM(1, 0,holdingRegs[angle2]);
  pwm.setPWM(2, 0,holdingRegs[angle3]);
  pwm.setPWM(3, 0,holdingRegs[grib]);
 // pwm.setPWM(0, 0,500);
```

Figure 22: Main code



```
if (holdingRegs[seq] == 1)

{
  for(int i = 0 ; i<= 4 ; i++){
    holdingRegs[angle1] = 100;
    holdingRegs[angle2] = 280;
    holdingRegs[angle3] = 240;
    holdingRegs[grib] = 300;
    up_date();
    delay(2000);

    holdingRegs[grib] = 100;
    up_date();
    delay(1000);
    holdingRegs[angle3] = 160;
    holdingRegs[angle2] = 100;
    holdingRegs[angle1] = 500;
    holdingRegs[seq] = 0;
     up_date();
     delay(1000);
     holdingRegs[grib] = 300;
    up_date();
    delay(4000);
  }

 }
```

Figure 23: Recorded sequence (motion)

Viii. Graphical User Interface (GUI)

This GUI is designed with InduSoft web studio, all motion control signals come from the GUI (Desktop Application) through MODBUS RTU communication protocol.

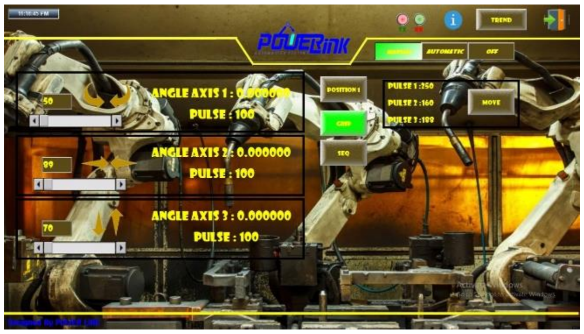


Figure 24: GUI

In this program we have a communication sheet which includes the addresses of the required registers to be accessed on the microcontroller also we specify the required communication process (Writing & Reading).

GUI symbols

- Three slide bars each one control an axis of the robot.
- Textbox + Arrows to control the robot axes step by step.
- Grip button to open and close the robot gripper.
- SEQ button to perform the pre-programed sequence of motion which written in the Arduino program.

Robotic Arm Operating Modes

There are 4 modes of operation

1. Continuous mode using slide bar
2. Step mode using textbox + Arrow
3. Move mode
4. Pre-programmed sequence

iX. Resources

Mechanical design source:

Web Link

Source Code:

Web Link

Bill of material:

Web Link

X. Conclusion

Finally, in this project we increase our technical and laboratory skills after implementing this project which is considered one of the most important project in industry field and we know how to integrate between the mechanical system and the electrical system and deal with programming code, in addition to gain more communication skills and team work and we looking forward to add some new features to this project and apply more related control projects.