Andrix Industry Sample Usage

Practical Robotic Institude Austria
TGM - The Institute of Technology Austria

2015 May

Kevin Pan

Alexander Wurm

Practical Robotics Institute Austria

tgm Die Schule der Technik

Contents

Part I
Introduction

1 About

1.1 PRIA

The Practical Robotics Institute Austria (PRIA) is a non-profit organisation with the aim to promote scientific and technical excellence in schools using robotics as well as the participation and operation of exemplary research projects in fields related to robotics and automation and teaching methods by means of robotics. The Practical Robotics Institute Austria is constituted as an independent non-profit organisation with a scientific advisory board.

Robots are sophisticated, intelligent systems used in factories and everyday life. They allow students to connect theory with practice and exercise team work, project management, problem solving and communication skills in a stimulating setting. Robotics integrates all the skills needed for designing and constructing machines, computers, software, communications systems and networks. It conveys to students the flexibility needed for developing interdisciplinary projects and discovers exciting topics such as movement, navigation, coordination, physical interaction with objects, audio and video processing, cognition and recognition and many others. It also helps them in developing their abstract thinking and to acquire teamwork skills, independence, imagination and creativity. Experiments with robots include many hands-on experiments that make classes more dynamic and fun.

1.2 Our Mission

- To use robotics technology as an instrument to prepare students to work with various mechanical, electrical and software systems.
- To develop environments and tools for teaching robotics that
  - will enable simpler programing and control of robots,
  - are easily interfaced with a broad range of sensors and effectors,
  - can be used to investigate complex problems and tasks,
  - are affordable and open for public use.
- To introduce appropriate educational methodologies for the successful integration of robotics innovation in school classes.
- To organise robotics competitions and measure their educational impact.
- Create innovative control architectures for robotics and industrial automation.
- Develop knowledge-based systems and agent technology for the automation of flexible industrial processes.
- Introduce Augmented Reality and make visualization of industrial processes more efficient and easier to understand.

2 Concept

2.1 Idea

Nowadays computers with ARM architecture are becomming more common. Various smartphones, as well as sigle-board computers like Raspberry Pi are becomming cheaper and much more affordable. To be able to use ARM Devices as robot controllers, some additional hardware that enables connecting and controlling robot components such as sensors, motors and servos are required. The hardware (aka. Low-Level Control, LLC) and the ARM devices(aka. High-Level Control, HLC) should be able to be connected together using both wired and wireless communications.

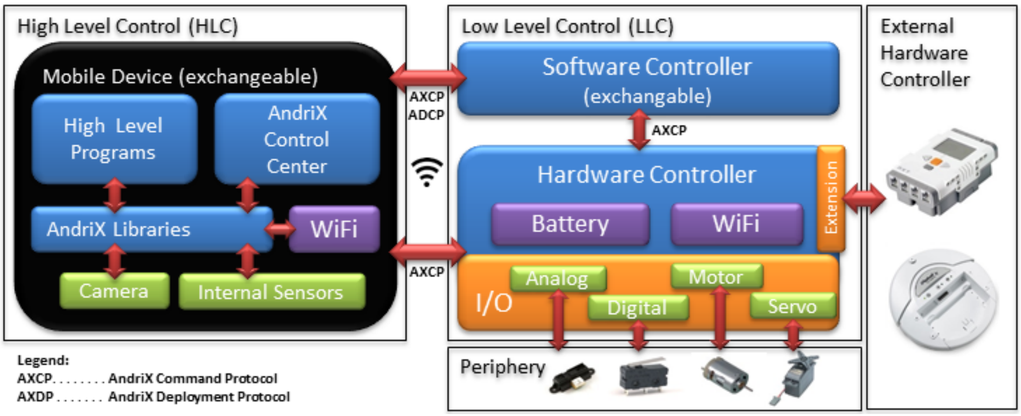


Figure 1: Andrix Concept

Andrix Industry is the concept of implementing this kind of structure in an industrial environment and its controller has to be relatively low-cost, stable and expandable for different uses.

In this project, we use a standard industrial conveyor belt to test our idea. We will connect our Andrix controller (including an Raspberry Pi as the High level control) to an external Motor Controller (aka. External Hardware Controller level) to be able to drive the conveyor belt using a Stepper Motor.

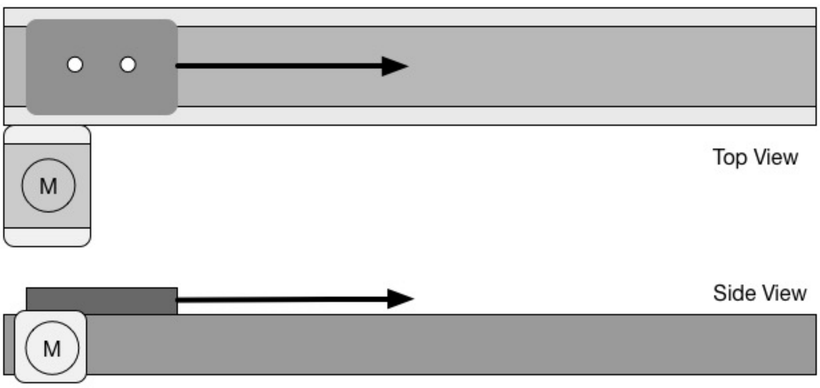


Figure 2: conveyor belt with NEMA stepper motor

THe NEMA Stepper motor series is one of the most popular motors in both amateur and professional fields. They are amazingly accurate and powerful, but at the same time very easy to control.

3 Process

3.1 Project steps

To be able to achive this goal, we planned the project precisely. The following provides a brief overview of our processes:

1. Choosing matching hardware
2. Designing necessary mechanical parts
3. Producing mechanical parts by using CNC or 3D-printer
4. Calculating needed supply power
5. Designing external motor controller
6. Prototyping electronic design
7. Communication protocol
8. Software programming
9. Debug, test and other final process

There are some different conditions which have to be satisfied to achieve our concept. At first, we need to find an industrial conveyor belt, which has to be samll enough to let us drive it easily with a stepper-motor without having too many problems. There are mainly two kinds of conveyor belt on the market today - limited and unlimited countineus conveying systems.

3.2 Selecting a suitable conveyor belt

Unlimited countineus conveying systems are the most common version and have been applied in various areas like raw meterial supply or warehouse transportation systems. They don't need any sesors to detects current position and can be controlled easily by changing the speed of the rotations. Limited conveyor belts work in principle familiar to continues system, but the position of the belt is controlled precisely use servo motor and has integrated sensors working with Hall effect(magnet) or IR encoders(light), but both of them need an extra circuit to drive them and need an externel programme to decode the position.

3.3 Selecting actuator

This is also one of the reasons why we use a stepper motor. NEMA stepper motors are in comparision to other motors much more flexible. It can be controlled easily with two H-Bridge Motor controller or FET-Transistors. We can dettect the position just by counting the steps in the register from the microcontroller. Of course it also has a disadvantage - speed. The accurency and the speed of the motor are proportional to each other and in our case, this can lead to inaccuracies by changing the step resolution, which is not a mechanical problem that can be easily avoided.

3.4 Communication

An agreement for the communication protocol between the Andrix and the external motor controller has to be made too. We are using 9600 bit/s as our communication speed. It is a standard bandwidth and can be controlled by other computers. The electronic controller has to be choosed carefully. We have to calculate and test the maximum supply current the motor need to run stably and compare it with the datasheet from the manufactuer.

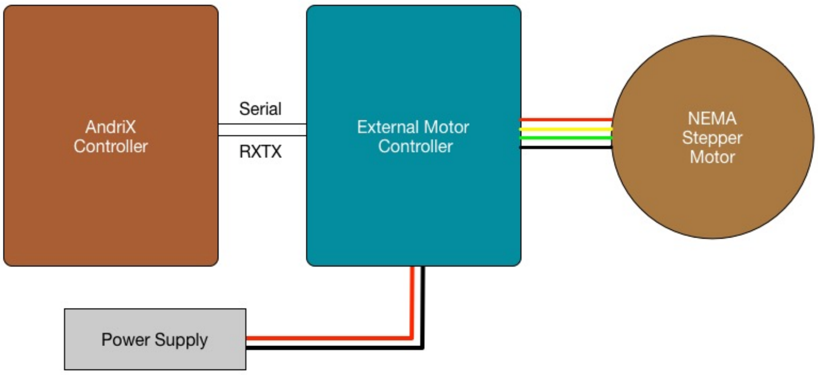


Figure 3: Control Flow

Part II
Design

4 Mechanics

4.1 conveyor belt and Axis

For the conveyor belt, we chose the Bosch REXroth CKR linear motion series. The CKR Compact Modules are precision, ready-to-install linear systems offering high performance within a compact envelope and selectable length, combined with an economical price-to-performance ratio.

Construction:

- Extremely compact precision aluminum profile with two integrated ball rail guides for optimal movement of heavy loads at high speeds
- Ready-to-install Compact Modules in selectable lengths up to Lmax
- Short or long aluminum carriage lengths available, depending on application load
- Driven by a pre-tensioned, steel-cord reinforced polyurethane toothed belt

Optional Components:

- Maintenance free digital servo drives with integrated brake and pre-installed feedback
- Gear Reduced type LP
- REED or HALL sensor switches available
- Socket with terminals for connection switches
- Aluminum profile cable duct for easy mounting of switches

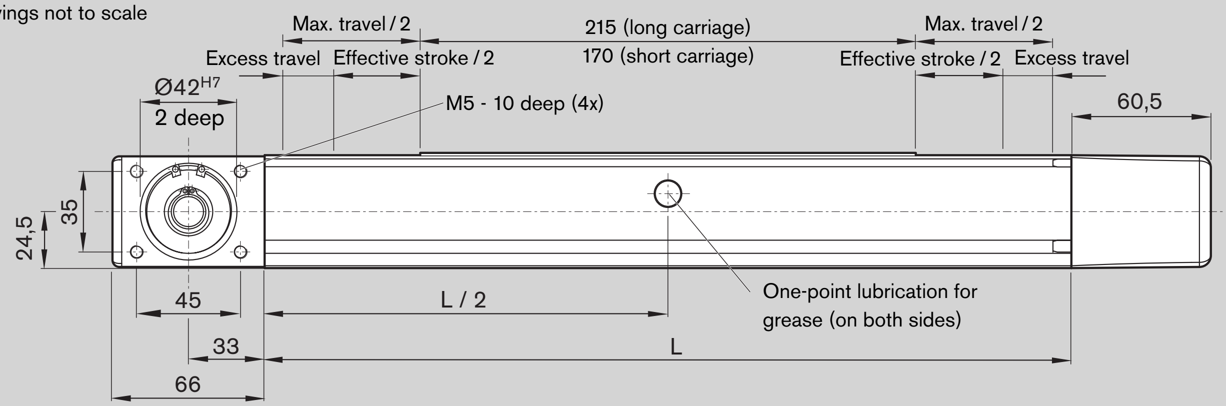
Capabilities:

- Speed - up to 5 m/s
- Load Capacity C - up to 56,530 N
- Length - up to 5,500mm
- Height - 40mm to 65mm
- Static Loading - up to 200kg

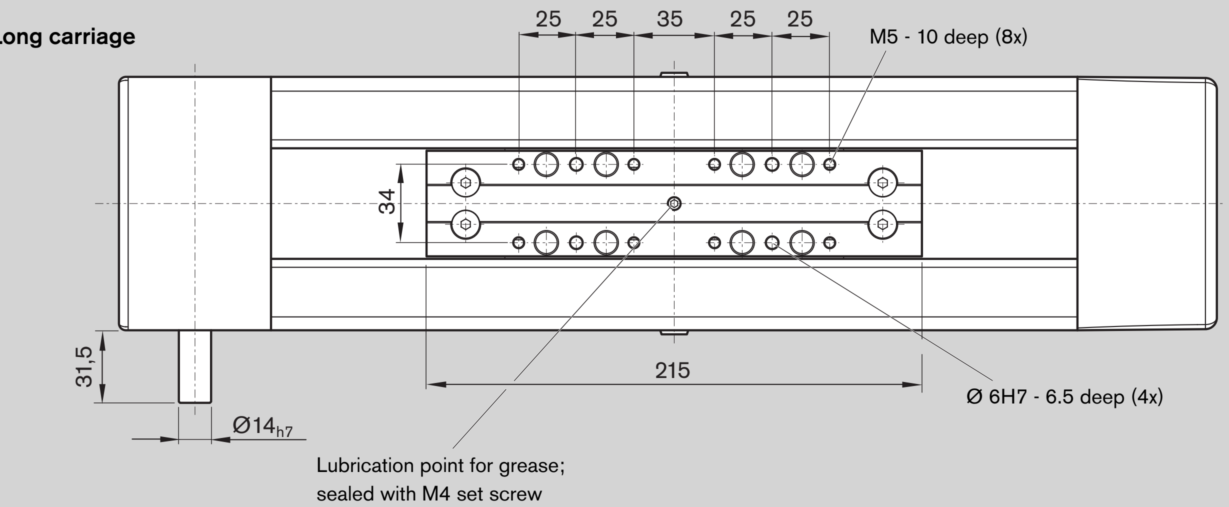
Compact Modules CKR

CKR 15-110 dimensions

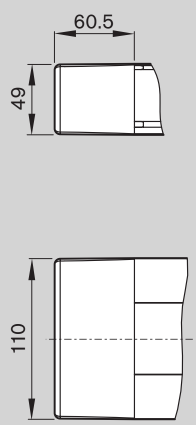
All dimensions in mm
Drawings not to scale



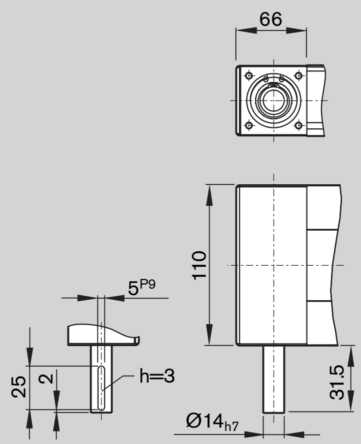
Long carriage



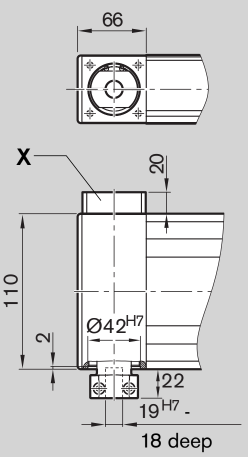
Type OA01



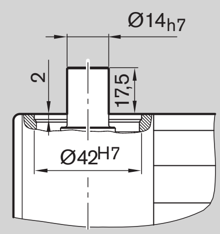
Types MA01 and MA02

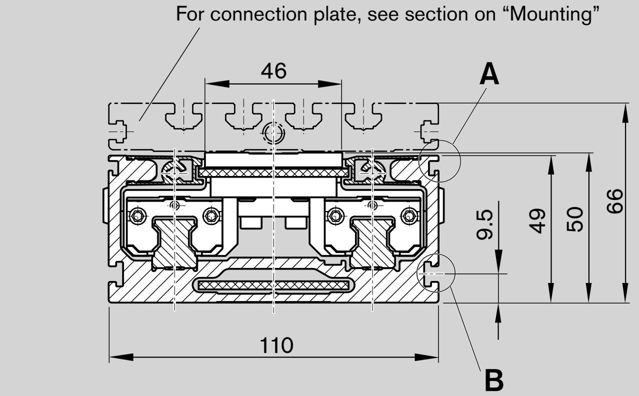


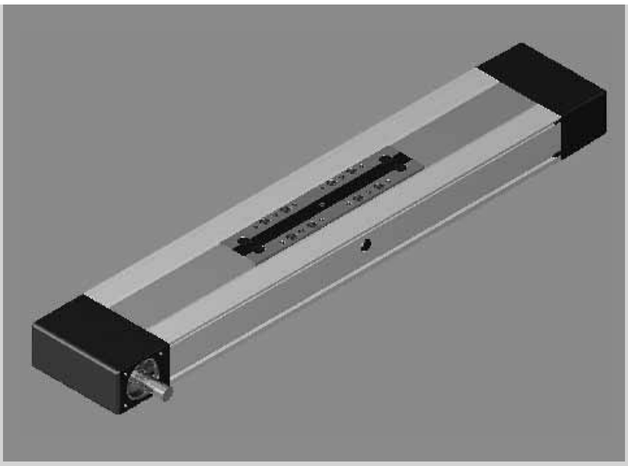
Types MA05 and MA06



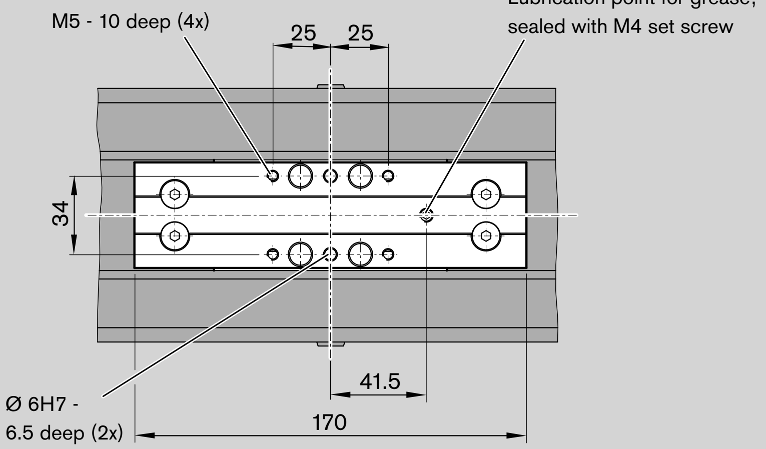
X

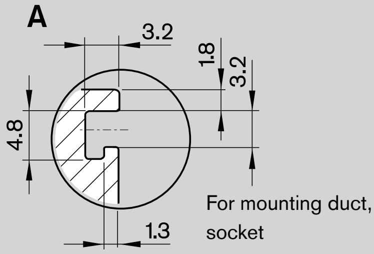


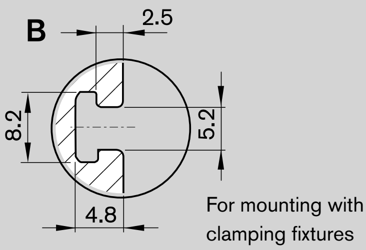




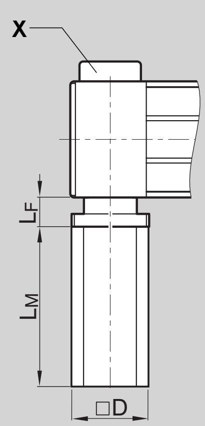
Short carriage



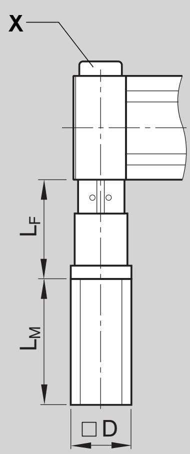




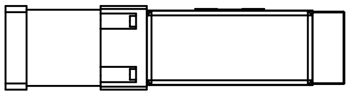
Types MA10 and MA11

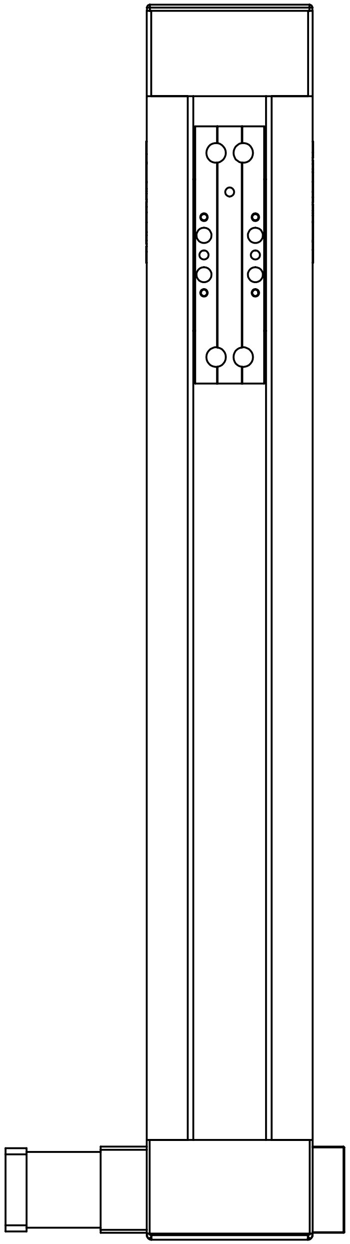


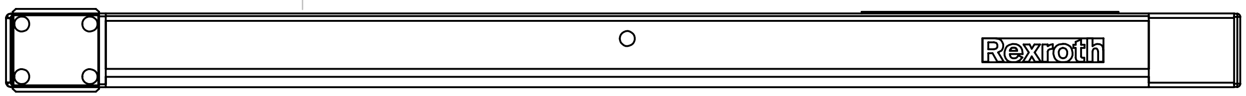
Types MG10 and MG11

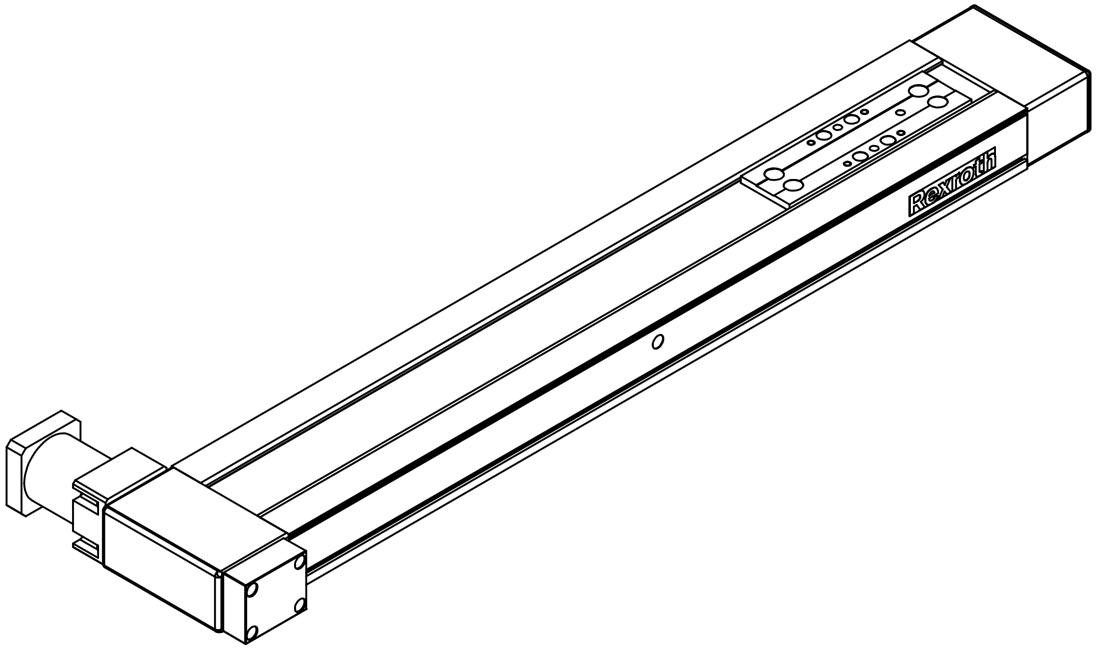


Type	Motor	Dimensions (mm)			
		D	L_F	L_M	
				without brake	with brake
MA10 MA11	MSK 050C	98	46.0	203.0	233.0
MG10	MSK 030C	54	93.5	188.0	213.0
MG11	MSM 030C	60	93.5	138.5	171.5









WENN NICHT ANDERS DEFINIERT:
BEMASSUNGEN SIND IN MILLIMETER
OBERFLÄCHENBESCHAFFENHEIT:
TOLERANZEN:
LINEAR:
WINKEL:

OBERFLÄCHENGÜTE:

ENTGRATEN UND SCHARFE KANTEN BRECHEN

ZEICHNUNG NICHT SKALIEREN

ÄNDERUNG

	NAME	SIGNATUR	DATUM
GEZEICHNET			
GEPRÜFT			
GENEHMIGT			
PRODUKTION			
QUALITÄT			

BENENNUNG:

WERKSTOFF:

GEWICHT:

ZEICHNUNGSNR.

CKR_110_NN_1_69

A4

MASSSTAB:1:1

BLATT 1 VON 1

4.2 Actuator and motor

The motor we are using is QSH5718-76-28-189 from TRINAMIC. It is a two phase hybird stepper motors optimized for microstepping and gives a good fit to high accurate control.

Specification:

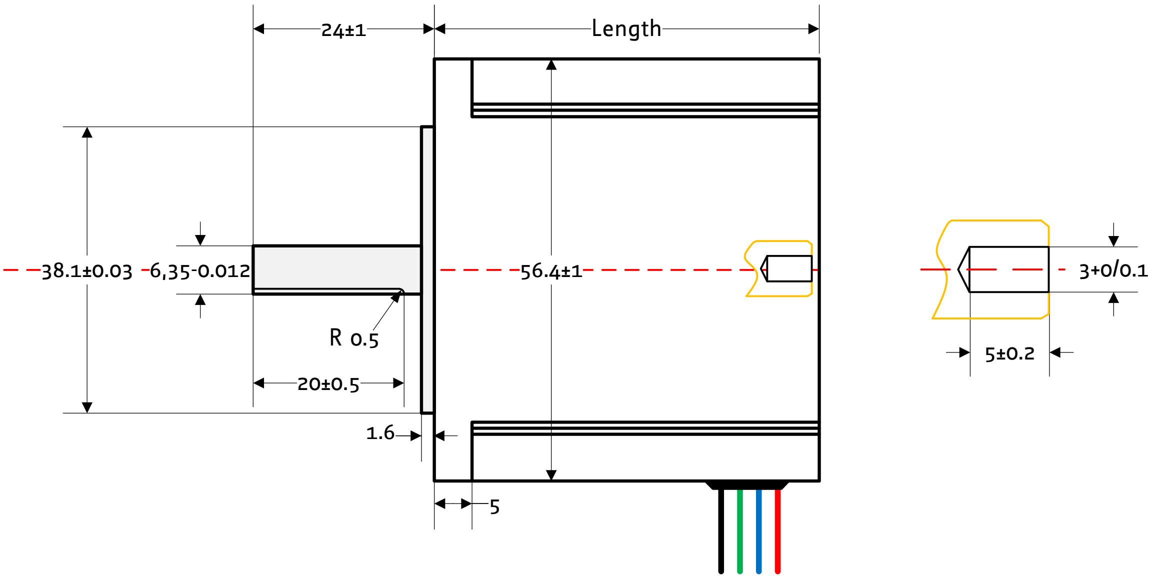
- NEMA 23 mounting Configuration
- Number of Leads: 4
- Step angle: 1.8
- Step Angle Accuracy: 0.05
- Holding Torque: 1.89 Nm
- Max app. Voltage: 75V
- Max Current: 2.8A /Phase
- Max Temp. Rise: +80
- Weight: 1kG
- Dimensions(mm): 56.5 x 56.5 x 76

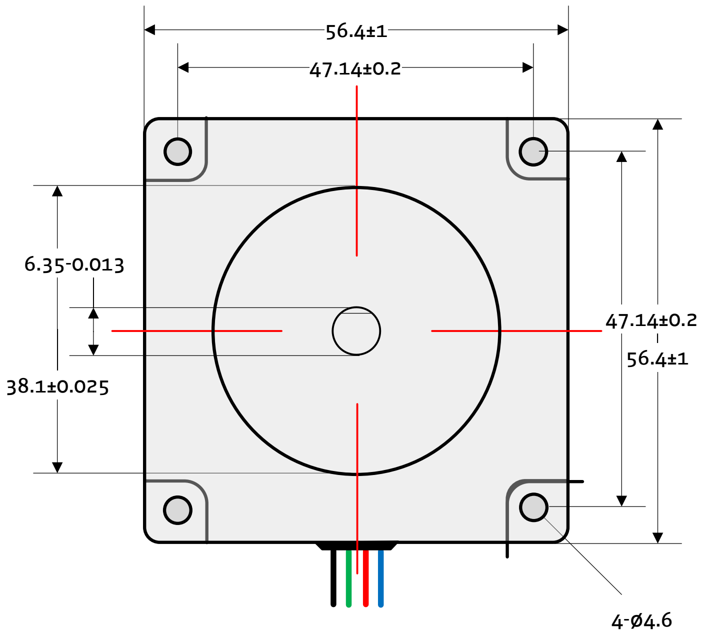
Specifications	Parameter	Units	QSH5718			
			-41-28-055	-51-28-101	56-28-126	-76-28-189
Number of Leads		N°	4	4	4	4
Step Angle		°	1.8	1.8	1.8	1.8
Step Angle Accuracy		%	5	5	5	5
Rated Voltage	V_{RATED}	V	2	2.3	2.5	3.2
Rated Phase Current	$I_{RMS\ RATED}$	A	2.8	2.8	2.8	2.8
Phase Resistance at 20°C	R_{COIL}	Ω	0.7	0.83	0.9	1.13
Phase Inductance (typ.)		mH	1.4	2.2	2.5	3.6
Holding Torque		Nm	0.55	1.01	1.26	1.89
Detent Torque		Nm	0.020	0.035	0.039	0.066
Rotor Inertia		g cm^2	120	275	300	480
Insulation Class			B	B	B	B
Max. applicable voltage		V	75	75	75	75
Max. radial force (20mm D-cut)		N	75	75	75	75
Max. axial force		N	15	15	15	15
Weight		kg	0.45	0.65	0.7	1
Length		mm	41	51	56	76
Temp. Rise (rated current, 2 phase on)		°C	+80 max	+80 max	+80 max	+80 max
Ambient Temperature		°C	-20 +50	-20 +50	-20 +50	-20 +50

Figure 4: Specifications of QSH5718 Series

4 Mechanical dimensions

4.1 Dimensions





Motor	Length (mm)
QSH5718-41-28-055	41
QSH5718-51-28-101	51
QSH5718-56-28-126	56
QSH5718-76-28-189	76

Figure 4.1: Dimensions of QSH5718. All values in mm.

4.2 Leadwire configuration

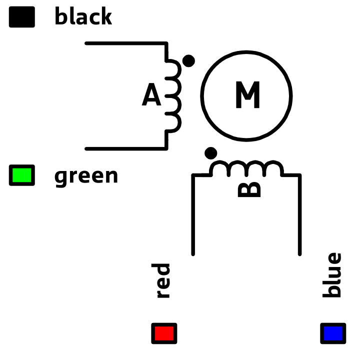


Figure 4.2: Leadwire configuration

Cable type 1	Gauge	Coil	Function
Black	UL1007 AWG22	A	Motor coil A pin 1
Green	UL1007 AWG22	A-	Motor coil A pin 2
Red	UL1007 AWG22	B	Motor coil B pin 1
Blue	UL1007 AWG22	B-	Motor coil B pin 2

5.4 QSH5718-76-28-189

VM: 30V, 2,8A/Phase

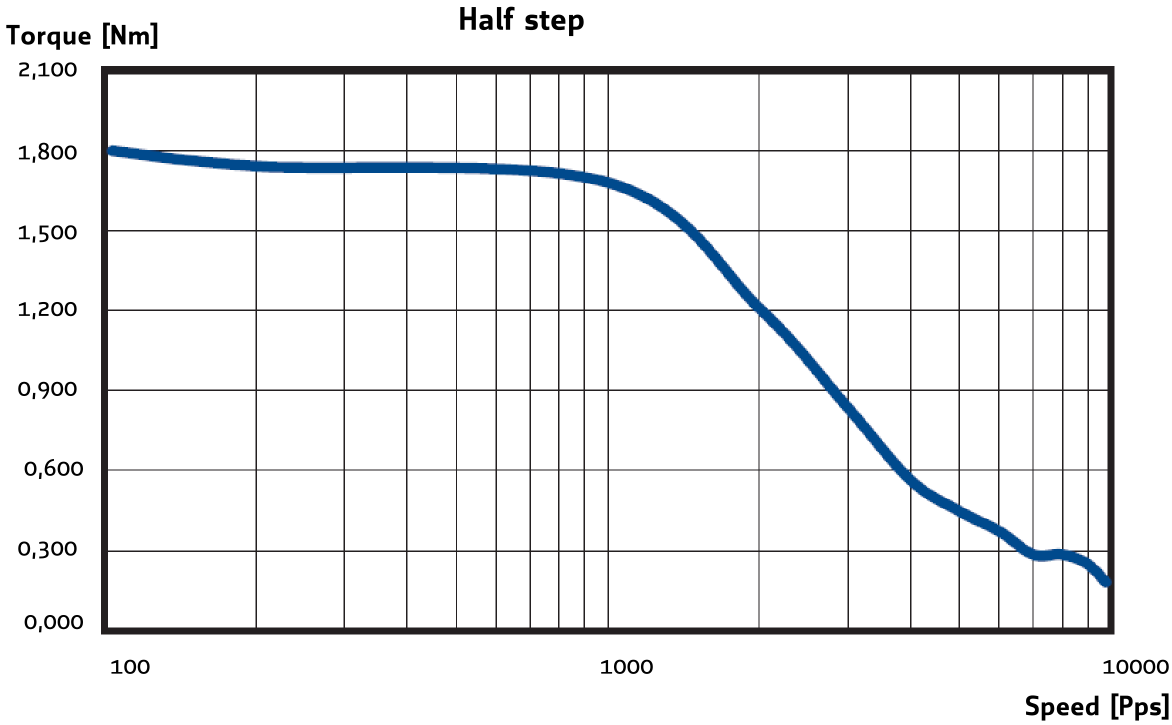
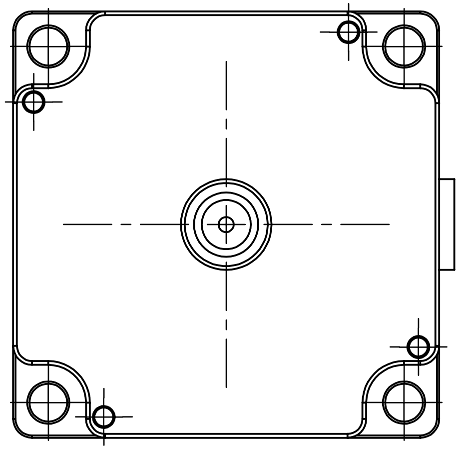
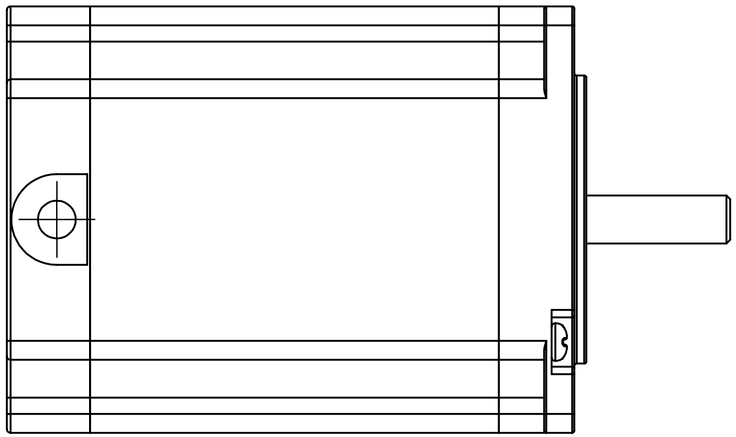
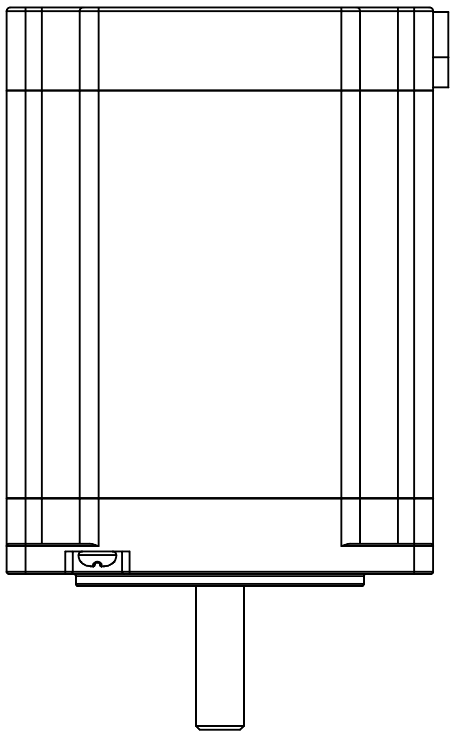


Figure 5.4: QSH5718-76-28-189 speed vs. torque characteristics

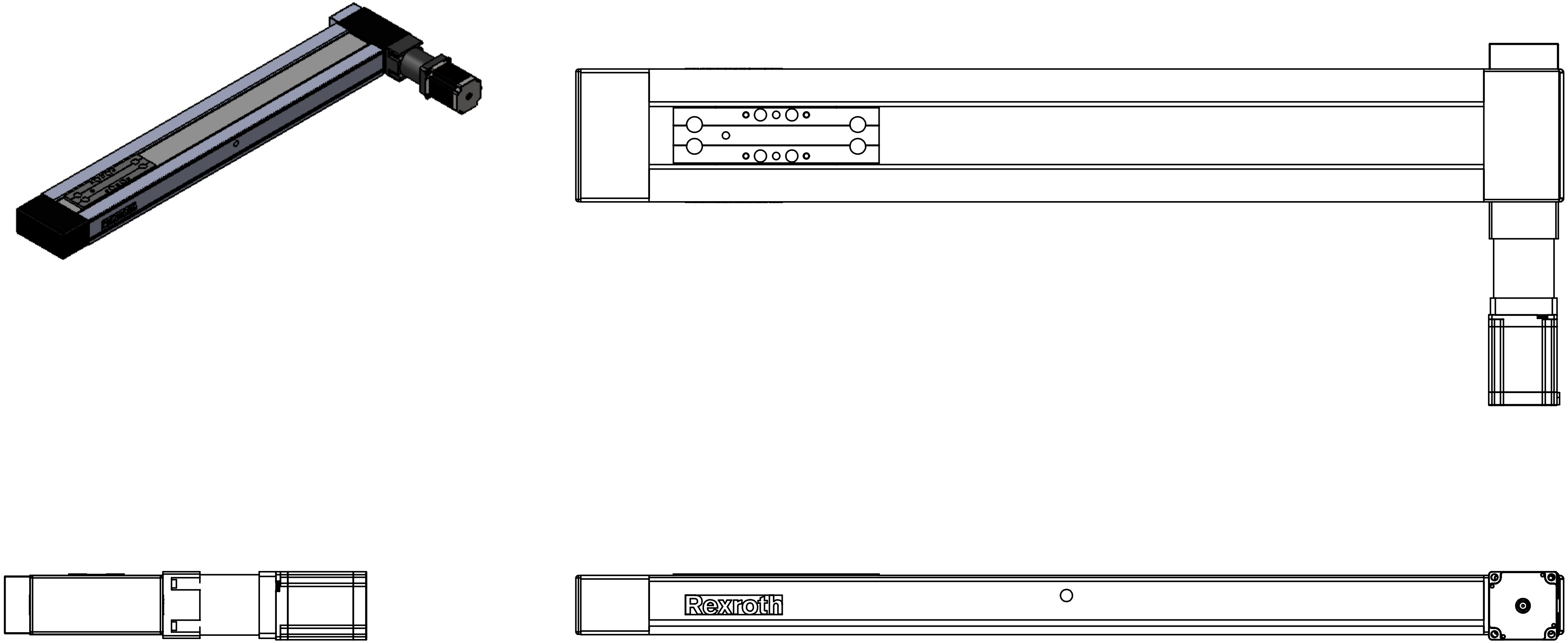






WENN NICHT ANDERS DEFINIERT: BEMASSUNGEN SIND IN MILLIMETER OBERFLÄCHENBESCHAFFENHEIT: TOLERANZEN: LINEAR: WINKEL:	OBERFLÄCHENGÜTE:				ENTGRATEN UND SCHARFE KANTEN BRECHEN	ZEICHNUNG NICHT SKALIEREN / ÄNDERUNG
	NAME	SIGNATUR	DATUM			BENENNUNG:
GEZEICHNET						
GEPRÜFT						
GENEHMIGT						
PRODUKTION						
QUALITÄT				WERKSTOFF:		ZEICHNUNGSNR. / A4
				GEWICHT:		MASSSTAB:1:1 / BLATT 1 VON 1

Stepper motor NEMA 23HS74



WENN NICHT ANDERS DEFINIERT: BEMASSUNGEN SIND IN MILLIMETER OBERFLÄCHENBESCHAFFENHEIT: TOLERANZEN: LINEAR: WINKEL:		OBERFLÄCHENGÜTE:			ENTGRATEN UND SCHARFE KANTEN BRECHEN	ZEICHNUNG NICHT SKALIEREN	ÄNDERUNG
	NAME	SIGNATUR	DATUM			BENENNUNG:	
GEZEICHNET							
GEPRÜFT							
GENEHMIGT							
PRODUKTION							
QUALITÄT				WERKSTOFF:		ZEICHNUNGSNR. CAD_Assembly	A4
				GEWICHT:		MASSSTAB:1:10	BLATT 1 VON 1

5 Electronics

5.1 External motor control unit

The external motor control unit consists of 2 different boards. One board is responsible for communication and translates the commands from the Andriox controler to the actual PWM signal and direction output. Another part of this unit is the motor controller. As the requirement, our stepper motor is bipolar and contents 4 wires. Two in a pair and each pair is only responsible for one coil.

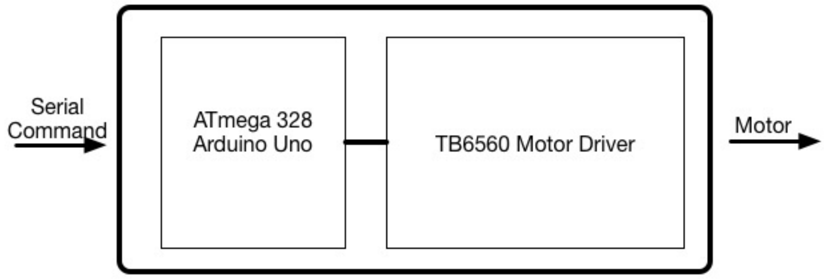


Figure 5: ECU Overview

For communication we decided to use a prototype board - Arduino uno. On the board is a ATmega 328 and has built-in serial communication pins. The standard Arduino IDE has pre implemented libraries for both serial and PWM signals.

The same as the describtion from the NEMA 23 Motor data sheet, we also found a stepper motor driver with required output current. The core of this board is a Toshiba dual H-bridge chip TB6560.

Model	QSH 5718-76-28-189
Max. Voltage	75 V
Rated phase current	2.8 A
Number of leads	4

Table 1: NEMA 23 requierment

Operating Voltage	12 - 36 V
Max. Output current	1.5 - 3 A/phase
Drive type	Double-pole const. PWM
Compatible Stepper motors	4/6/8 leads

Table 2: TB6560 Data sheet

5.2 Power supply

As we are suppplying our motor with 12V, the calculation of the power consumption will be like this:

$$\text{Total power } P_t = \text{External Control Unit } P_e + \text{Motor } P_m \tag{1}$$

$$P_m = I_o \times U_i = 3A \times 12V = 36W \tag{2}$$

$$P_m \gg P_e \text{ ,so } P_m \approx P_t \tag{3}$$

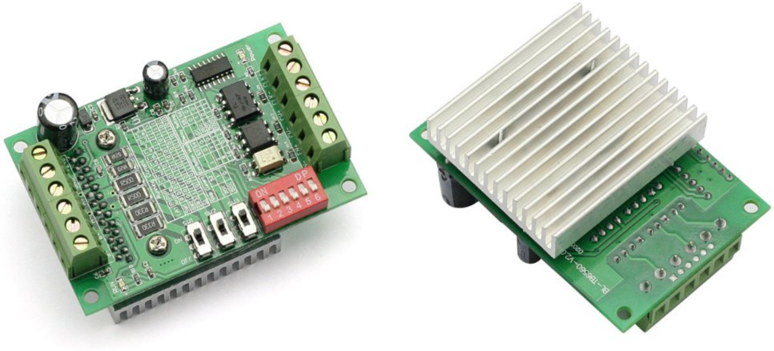


Figure 6: TB6560 Stepper Motor Driver Module

Our power supply must able to supply at least 40 Watts of power to drive our converying belt and control units.

5.3 Overall circuit

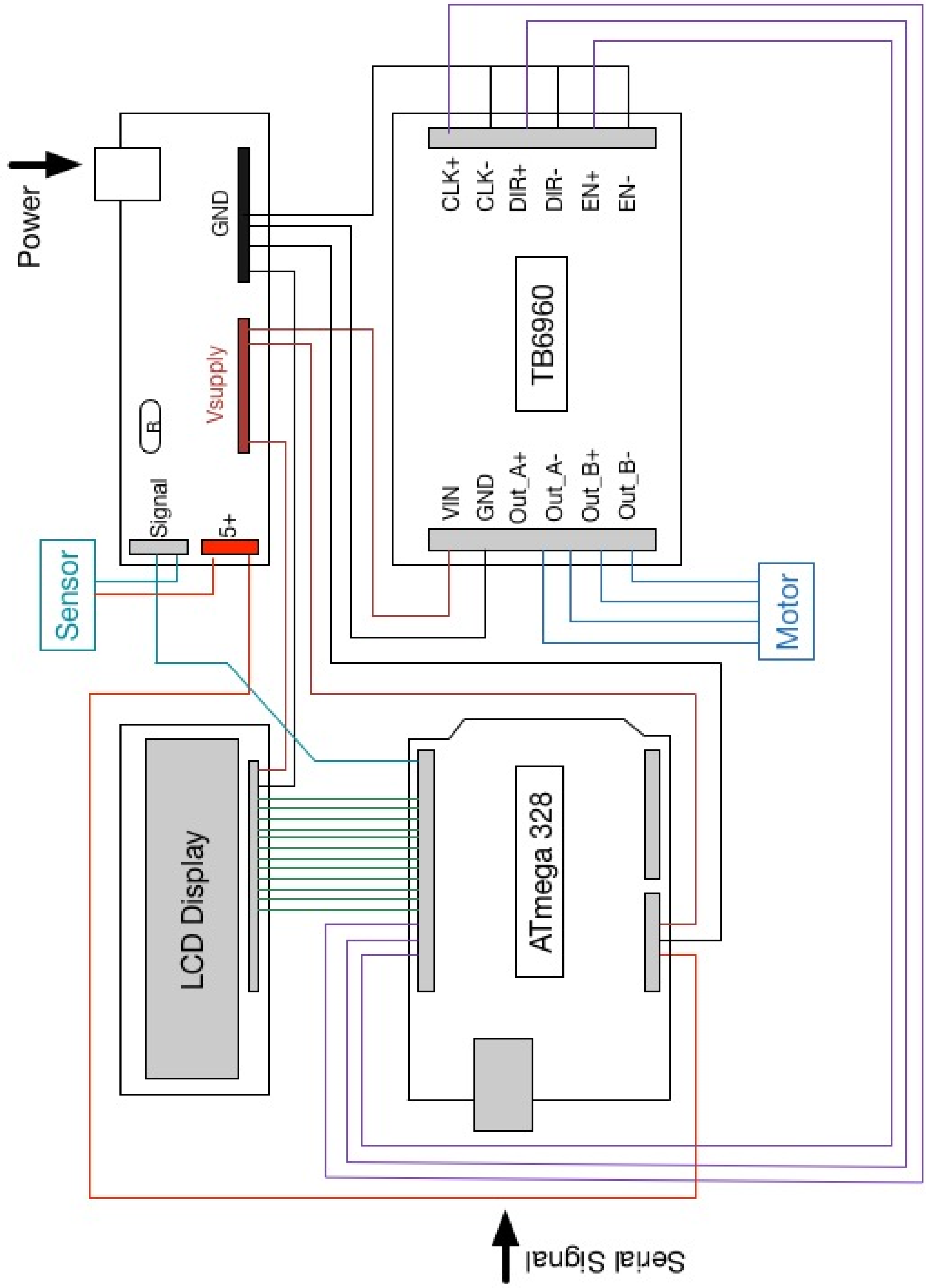


Figure 7: External Control Unit: cabling

6 Software - Ecu

6.1 Logical equivalence

The system logic of the external motor controller has 3 main states: initial, processing and waiting process:

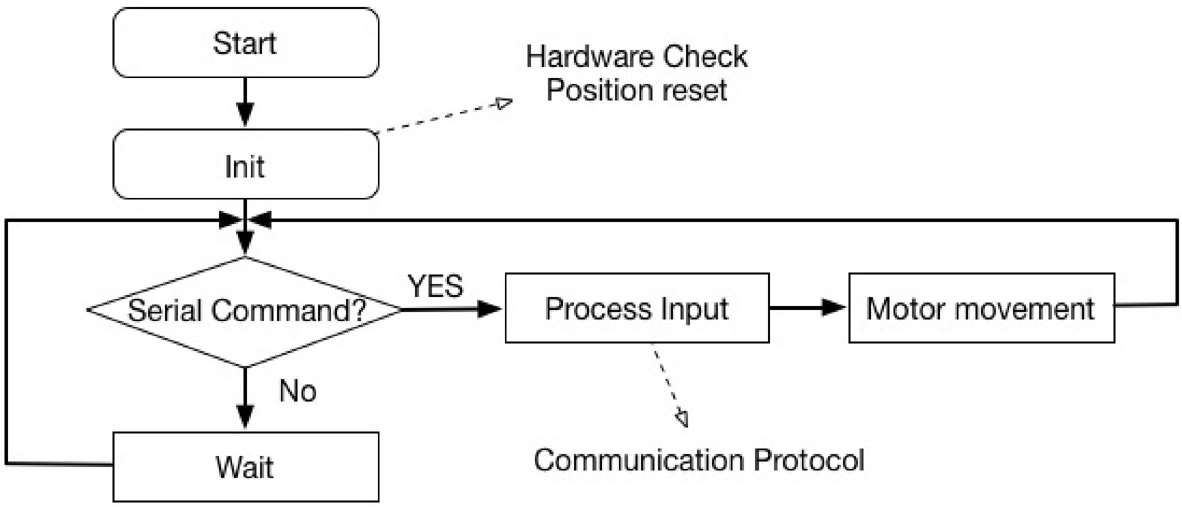


Figure 8: Software Logic

In the boot up phase, the system will first check if all the hardware is connected and reset the position automatically. It sends continuous signals to the motor to move the conveying base back to the origin and if the base has a physical contact with it, the digital sensor will set its output to HIGH.

After the reboot, the system is ready and waits for the serial command from the andrix or PCs. The serial bandwidth has already been set in the initial process and the default is 9600bit/s. Any serial signal during the waiting phase will be stored in a String buffer to futher process. The conmunication will be base on the protocol from **section 6.3**.

6.2 Function description

To find more detail of a specific function, please go to **section 6.4**.

Name	Use	Input	Return
setup()	Init Setup	—	—
loop()	Main loop	—	depend
serialEvent()	Reading Serial Input and buffer them	—	—
reset()	Reset position to origin	—	—
goToPosition()	Go to position	Integer	—
lcdShow()	Display message on LCD Display	String 1,2	—
lcdBackLightOnOff()	Turn LCD Light On or Off	boolean	—

6.3 Communication protocol

Command ...@	Description @ - end; ... - Commands	Sample Bandwith = 9600 bit/sec
ENA@	Enable the motor	Send: **ENA@** – Motor enabled
DIS@	Disable the motor	Send: **DIS@** – Motor disabled
RES@	Reset, back to origin	Send: **RES@** – Axis back to 0 position
GTP...@	Go to xxx/10 mm position [Max. 4950]	Send: **GTP1234@** – Axis go to 123.4mm
SST...@	Set delay of motor(default 0) [2 byte]	Send: **SST1@** – Set 1 ms delay for each step
CTP@	send back to the current position [2 byte]	Send: **CTP@**;Return: **123@** – Current Position
CTS@	send back the current speed(delay) [2 byte]	Send: **CTS@**;Return: **1@** – Current Delay
LED...@	Turn LED Display light ON or Off	Send: **LED1@**–Light turn On

Table 3: Protocol: Stepper Motor Control

If you want to modify the setting of the communication bandwidth, you will have to programme the external control unit with the source code from **6.4** using Arduino IDE.

Communication sample:

```
Normal Operation:
1.SEND:        ENA@           \\Enables the motor
2.SEND:        GTP1000@       \\Go to Position 10cm
3.SEND:        RES@           \\Reset position to 0, all default value


Move Slowly:
1.SEND:        ENA@           \\Enables the motor
2.SEND:        SST1@          \\Set Delay for 1ms
3.SEND:        GTP500@        \\Convayingbelt will move slowly to 5cm


Let conveyor belt to be able to move by hand:
1.SEND:        DIS@           \\Disable the motor
2.SEND:        RES@           \\WARNING: You must reset the position!


Turn On/Off LED Display backlight:
1.SEND:        LED1@          \\Turn On light
2.SEND:        LED0@          \\Turn Off light
3.SEND:        LED3@          \\Turn On light(Any num.>0 will be 'ON')
```

6.4 Source code

```
/*
  Name:            Stepper Motor Controller (Serial Communication) - 3.5A low power
  Autor:           Alexander Wurm, Kevin Pan
  Discribtion:     Stepper Motor Controller (Serial Communication) - 3.5A low power
  Last Update:     04.2015
  Contact:         kpan@student.tgm.ac.at, awurm@student.tgm.ac.at
*/
#include <LiquidCrystal.h>
//Pin setups:
#define Sensor_Touch_Sensor_Pin 7       //Touch Sensor(init) pin
#define Motor_Clock_pin 9               //Clock(Tacks) CLK+ pin
#define Motor_Direction_pin 8           //Direction CW+ pin
#define Motor_Enable_pin 10
#define Lcd_D7_pin 2
#define Lcd_D6_pin 3
#define Lcd_D5_pin 4
#define Lcd_D4_pin 5
#define Lcd_Enable_pin 11
#define Lcd_RS_pin 12
#define Lcd_Backlight_pin 6

//Static values:
#define Maximum_steps 32250             //Maximum steps
#define Axis_length 4950                //Maximum Length
#define Clock_impuls 100                //Microsecond

//Variables:
int Current_position = 0;               //Milimeter
float Step_resolution = 0.15349;        //Stepmotor step-resolution in mm
String inputString = "";                // a string to hold incoming data
boolean stringComplete = false;         // whether the string is complete
int delayOfMotor = 0 ;
boolean motor_enable;                  // Remembers the status of motor en/disable
LiquidCrystal lcdDisplay(Lcd_RS_pin, Lcd_Enable_pin, Lcd_D4_pin,Lcd_D5_pin,Lcd_D6_pin

void setup() {
  //Lcd Display setup:
  lcdDisplay.begin(16,2);
  pinMode(Lcd_Backlight_pin, OUTPUT);
  lcdBackLightOnOff(true);
  lcdShow("System: Starting","Please Wait...");

  //Reserving memory spaces for inputStream:
  Serial.begin(9600);
  inputString.reserve(32);

  //Motor pin Setup:
  motor_enable=true;
  pinMode(Motor_Enable_pin,OUTPUT);
  pinMode(Motor_Direction_pin, OUTPUT);
```

```
  pinMode(Motor_Clock_pin, OUTPUT);
  pinMode(Sensor_Touch_Sensor_Pin,INPUT);
  digitalWrite(Motor_Enable_pin,LOW);
  digitalWrite(Motor_Direction_pin, LOW);
  digitalWrite(Motor_Clock_pin, LOW);
  delay(2000);

  //Start Reseting Position:
  lcdShow("Reseting  POS.","Please Wait...");
  delay(2000);
  reset();

  //Finish Reseting Position:
  lcdShow("Reseting  POS...","Finished...");
  digitalWrite(Motor_Enable_pin,HIGH);
  motor_enable=false;
  delay(1000);
  lcdShow("Waiting Command"," POS");
}

void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    inputString += inChar;
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '@') {
      stringComplete = true;
    }
  }
}

void loop() {
  if (stringComplete) {
    String command=inputString.substring(0,3);
    String showCommand="Command: "+command;
    lcdShow(showCommand," POS");
    if(command=="ENA"){
      //Serial.println("motor enabled");
      digitalWrite(Motor_Enable_pin, LOW);
      motor_enable=true;
    }
    else if(command=="DIS"){
      //Serial.println("motor disabled");
      digitalWrite(Motor_Enable_pin,HIGH);
      motor_enable=false;
    }
    else if(command=="RES"){
      //Serial.println("Position resets");
```

```
        reset();
      }
      else if(command=="GTP"){
        int pos=inputString.substring(3,inputString.length()).toInt();
        //Serial.print("Going to position: \t");Serial.println(pos);
        goToPosition(pos);
      }
      else if(command=="SST"){
        int del=inputString.substring(3,inputString.length()).toInt();
        //Serial.print("Set delay to: \t");Serial.println(del);
        delayOfMotor=del;
      }
      else if(command=="CTP"){
        String out = String(Current_position) + "@";
        Serial.println(out);
      }
      else if(command=="CTS"){
        if(delayOfMotor==0){
            Serial.println("0@");
          }
        else{
            String out = String(delayOfMotor) + "@";
            Serial.println(out);
          }
      }
      else if(command=="LED"){
        boolean onoff = inputString.substring(3,inputString.length()).toInt();
        lcdBackLightOnOff(onoff);
      }
      else{
        //Serial.println("SYSTEM: Serial input error");
        //Serial.print("Input:\t");
        //Serial.println(inputString);
      }

      lcdShow(showCommand,"_POS");
      // clear the string:
      inputString = "";
      stringComplete = false;
    }
}

void reset(){                         //RESET THE POSITION BACK TO 0 USING SENSOR
    if(motor_enable==true){
      digitalWrite(Motor_Direction_pin, HIGH);
      while(!digitalRead(Sensor_Touch_Sensor_Pin)){
        digitalWrite(Motor_Clock_pin, HIGH);
        delayMicroseconds(Clock_impuls);
        digitalWrite(Motor_Clock_pin, LOW);
        delayMicroseconds(Clock_impuls);
        delay(delayOfMotor);
```

```
    }
    Current_position = 0;
  }
}

void goToPosition(int x_position){
    if(motor_enable==false){return;}

    //checking if the input value is posible:
    if(x_position<0 || x_position>4950){
      return;
    }

    //calculating the difference:
    int diff = x_position - Current_position;
    if(diff>0){
      digitalWrite(Motor_Direction_pin, LOW);
    }
    else if(diff<0){
      digitalWrite(Motor_Direction_pin, HIGH);
      diff=-diff;
    }
    else {
      return;
    }

    //If the position is going back to zero:(will be repalced with a digital button)
    if(x_position==0){
      int k = (int)(Current_position/Step_resolution);
      for(int i=1;i<=k; i++){
        digitalWrite(Motor_Clock_pin, HIGH);
        delayMicroseconds(Clock_impuls);
        digitalWrite(Motor_Clock_pin, LOW);
        delayMicroseconds(Clock_impuls);
        delay(delayOfMotor);
      }
      Current_position =0;
      return;
    }

    int steps = (int)(diff/Step_resolution);
    for(int i=1;i<=steps; i++){
      digitalWrite(Motor_Clock_pin, HIGH);
      delayMicroseconds(Clock_impuls);
      digitalWrite(Motor_Clock_pin, LOW);
      delayMicroseconds(Clock_impuls);
      delay(delayOfMotor);
    }
    Current_position = x_position;
}

```

```
void lcdShow(String outputString1,String outputString2){
  //Cleaning the screen:
  lcdDisplay.clear();

  //First row:
  lcdDisplay.setCursor(0, 0);
  lcdDisplay.print(outputString1);

  //Second Row:
  lcdDisplay.setCursor(0, 1);
  if(outputString2 == " POS"){
    String out;
    out = "POS: " + String(Current_position);
    out = out + " mm";
    lcdDisplay.print(out);
  }
  else{
    lcdDisplay.print(outputString2);
  }
}

void lcdBackLightOnOff(boolean a){
  if(a==true){digitalWrite(Lcd_Backlight_pin,HIGH);}
  else if(a==false){digitalWrite(Lcd_Backlight_pin,LOW);}
}
```

Part III
Conclution

After assembling the mechanical design and putting all electronic components together, we devised a very simple system. We tested the axis accuracy and it is much more percise than we first expected.

7 Resaults and Comparison

In comparison with some of the industrial standard machines, our solution can be up to 10 times cheaper. Of course, the standard system can control/supply multiple linear or even other kinds of axises at the same time. But one of the biggest advantages of distributed system is flexibility.

Unter the same kind of hardware structure, we are able to connect multiple controllers with a central computer, and control them induvidually. It is also even posible to implement some kind of smart algorithm, which is designed spicificly for each individual enviroment.

In this example we only show one way of using an Andrix controller in Industrial area. It shows us the potential of distributed system and can be one of the most important keys for 'Industrial 4.0'. It will change the way we look at the normal rigid structure.