

# Robolink Motion Editor



## User Guide Programmer Guide

## Contents

Introduction.....	3
Installation.....	3
User Guide.....	5
A. Keyframe Editor.....	6
B. Configuration.....	6
C. Keyframe Grabber.....	7
D. Status Message Area.....	7
E. File Manager.....	7
F. Motion Sequence.....	8
G. Sandbox.....	8
List of Keyboard Shortcuts.....	9
Programmer Guide.....	10
PC Software.....	10
Microcontroller software.....	13
Extended mode.....	13
Command execution.....	13
Initialization.....	13
Motion sequence playback.....	13
Hardware Configuration.....	14
Connecting the Encoder.....	14
Preparing the controller.....	14
Compiling and uploading NanoJMotorControl.....	14
Microcontroller connections.....	15
Start button semantics.....	15
Microcontroller firmware.....	16

## Introduction

The robolink motion editor software was developed in a joint project between igus GmbH and the Autonomous Intelligent Systems group of University Bonn. The purpose of the motion editor is to provide an easy to use graphical user interface for creation and playback of motions for the igus robolink prototype. The target operating system is Windows XP. The following document provides an overview of the available features starting with the installation of the software. Then in succession from top to bottom the graphical user interface is described, the application software components are sketched and finally, remarks to the NanoJ microcontroller program are given.

## Installation

The robolink motion editor does not need to be installed. It is provided as a zip file with the name *IgusMotionEditor.zip*. Unpacking the zip file creates an *IgusMotionEditor* folder that contains all necessary files to run the program. Use the executable *IgusMotionEditor.exe* to start. The motion editor can also be run from a USB stick by plugging the USB stick into any PC with a Windows XP operating system and starting the executable directly from the USB stick.

For the communication between the software and the robot, a virtual COM port driver is needed. The motion editor can be started without the driver and used for offline motion editing, but to connect the software with the robolink robot arm, the installation of this driver is required. The zip file contains the installation file of the Silabs CP210x driver for communication with the microcontroller board.

Please note that using the zip file to distribute the motion editor cannot include motions that were created later on with the software. It is possible, however, to transfer motions between computers by simply copying the contents of the *motions* folder, where the created motions are saved as .txt files.

The *calibs* folder contains a *robot.ini* file that describes the used robot. It consists of Joint specifications in a standard INI format:

```
[Joint0]
name=X1                # Displayed name
type=X                 # Joint type (X or Z)
address=1              # Motor controller address
lower_limit=-1.0       # Lower joint angle limit in radians (-π to π)
upper_limit=1.0        # Upper joint angle limit in radians (-π to π)
offset=0.0             # Joint offset in radians (-π to π)
encoder_steps_per_turn=3000 # Encoder steps per full joint turn
motor_steps_per_turn=6000  # Motor steps per full joint turn
invert=1               # Invert the axis (0 or 1, optional)
length=0.10           # Length for display in m (optional)
joystick_axis=0        # Axis on the joystick/gamepad (optional)
joystick_invert=1      # Invert the joystick axis (0 or 1, optional)
```

The joint specifications have to be ordered (Joint0, Joint1, Joint2, ...) in the order they appear on the robotic arm from base to gripper.

A global configuration section is also included with parameters for the whole robot arm:

```
[global]
lookahead=300                                # Velocity control lookahead in ms
```

The lookahead parameter can be used to fine-tune the velocity control algorithm for playback. A higher value results in more smooth starts and stops, while a value of zero disables the velocity control algorithm entirely. The user can use the features of the Nanotec controller in this case.

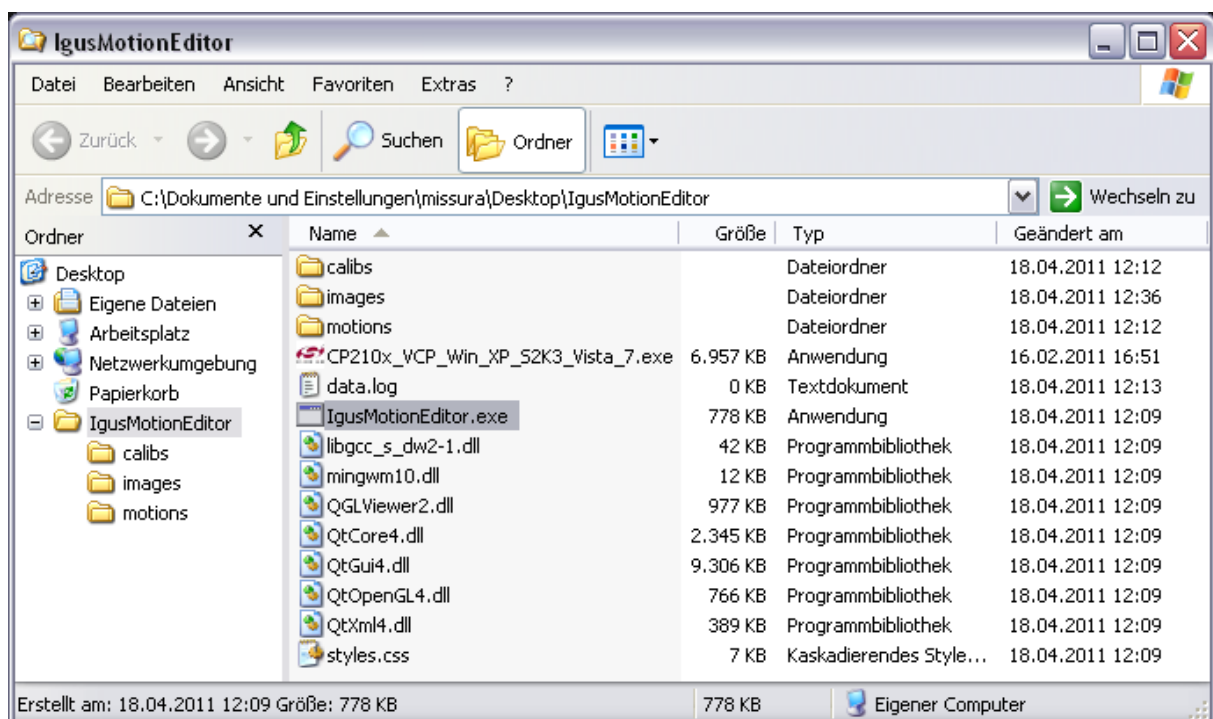


Figure 1: The relevant files and folders of the motion editor.

## User Guide

The graphical user interface can be divided into seven functional units denoted with letters A through G, as shown in Figure 2. In the following, these letters will be used to refer to the functional units.

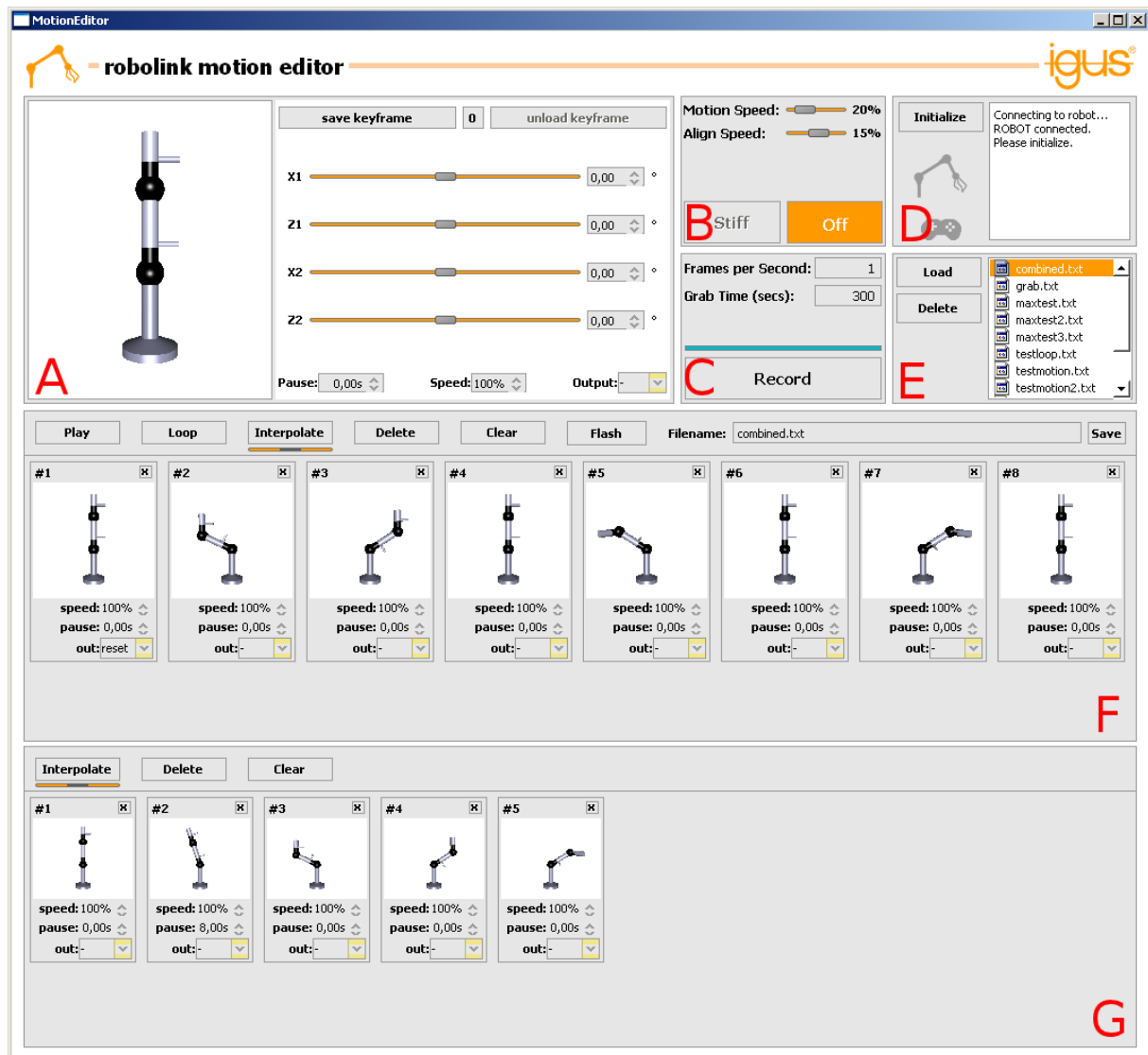


Figure 2: The graphical user interface.

Motions are defined by sequences of keyframes. One keyframe specifies a pose of the robot. When executing a whole motion, intermediate positions between keyframes are linearly interpolated and sent to the robot with a frequency of approximately 12 Hz.

The speed of the motion is controlled with a speed parameter that can be set for every keyframe. The speed parameter is a percental value of the maximum possible speed of the robot arm and defines how fast the robot is moving towards the keyframe. When the keyframe is reached, the speed of the next keyframe is applied. Please note that when moving towards a keyframe, not all joints rotate with the same

speed. The motion playback algorithm calculates individual joint velocities such that every joint reaches the position defined by the next keyframe at the same time.

Additionally, a pause parameter can be used to set the amount of seconds the robot will wait after a keyframe has been reached before continuing the motion.

The digital output of the microcontroller can be set, reset, or left as it is (default) at the time a keyframe is reached.

## A. Keyframe Editor

The letter A marks the keyframe editor component. Here, using the sliders or the spin boxes on the right, the joint angles of a single keyframe can be adjusted and fine tuned. A double click on any keyframe in the motion sequence area F or the sandbox G will load a keyframe into the keyframe editor. The sliders and spin boxes will be set to the joint angles stored in the loaded keyframe. If now the sliders are moved, the keyframe will be overwritten immediately. The function of the *save keyframe* button (shortcut key 'Enter') is to create a new keyframe with the joint angles currently set in the keyframe editor. The newly created keyframe will be placed in the sandbox G. The *unload keyframe* button can be used to unload a keyframe from the keyframe editor so that changes of the sliders will not be written into a keyframe any more. The '0' (zero) button resets all sliders to 0 and a double click on a slider resets that individual slider to 0. These operations also overwrite the currently loaded keyframe.

In the top left corner an interactive 3D model shows the current position of the robot. To inspect a robot pose, the model can be rotated while holding the left mouse button and panned while holding the right mouse button. The mouse wheel zooms the 3D view. A double click on the model resets it to the default view. Using clicks with the middle mouse button, individual joints can be selected in the model and then moved with the left mouse button pressed down. This is the same operation as moving the sliders. To deselect a joint, click the middle mouse button in the white area of the model.

## B. Configuration

In the configuration area B, sliders can be used to adjust the global velocity parameters of the motion editor. The Motion Speed slider defines how fast the robot is moving during playback at 100% speed. Using this slider will affect the playback of all previously saved motions. While the Motion Speed slider defines the global reference speed for all motions, the speed parameters of individual keyframes define local motion speeds (e.g. for one keyframe) with respect to this global setting. The motion speed is only used when the motion playback is activated using the play button (see area F). During motion editing, for example when double clicking and loading a keyframe, the robot moves with the slower Align Speed that can also be adjusted with the second slider, respectively.

The *Stiff* and *Off* buttons change the operation mode of the robot. In *Stiff* mode the robot is fully actuated and makes every attempt to take a commanded pose, even if it is hindered by external forces. In the *Off* operation mode the power to the joints is turned off. The *Off* mode can be used to test motions without actually moving the robot and also to move the robot into a new pose by hand.

## C. Keyframe Grabber

The functional unit denoted as C is a keyframe grabber that can be used to create keyframes with poses recorded directly from the robot. Use the frames per second and grab time parameters to determine the frequency and the total time of a record operation. The recorded keyframes will be appended to the sandbox area G. The enter key can be used any time to grab a single keyframe from a connected robot. To teach-in a motion, the robot can be set into Off mode, moved into a pose and a keyframe can be recorded by pressing the enter key. Repeating this procedure with several poses will quickly produce a set of keyframes and create a motion. Alternatively, the automated keyframe grabber can be started using the Record button and the robot arm can be moved by hand or with the joystick. When grabbing keyframes from a moving robot, the keyframe grabber will attempt to estimate the current motion speed and will automatically set the speed parameter.

## D. Status Message Area

Most of the motion editing functionality is available without the robot arm being connected and/or switched on. Motions can be created, edited and viewed offline before being tested on the real robot. To execute motions on the robot, plug in the USB cable of the virtual COM port adapter and switch the robot on. The software will automatically recognize the connection. Connection status messages are shown in the message area in the top right corner of the user interface (D). Before the robot can be used, an initialization procedure has to be completed during which the robot finds its own zero position using the mounted HALL sensors. The initialization procedure is triggered by pressing the Initialize button or the shortcut key 'I'. The robot arm icon left of the status message box turns orange to indicate a successful initialization of the robot. The robot will remember its initialization until it is turned off, so that it's possible to close the motion editor and to even unplug the robot and to continue the session later without having to initialize again.

Similarly, the joystick icon turns orange when a joystick (i.e. game pad) is connected to the PC. Except during motion playback, the robot can be moved any time using the joystick. Moving the joystick is essentially the same as moving the sliders in the keyframe editor A. If the robot is not connected, the joystick still moves the 3D simulation model.

**Note:** In order to use the joystick, the "joystick\_axis" Parameter in robot.ini needs to be configured (see above).

## E. File Manager

The area with the letter E is a miniature file manager that handles the saved motions. Motions saved with the Save button in area F appear in this list. A double click on a motion file or a click on the Load button appends the keyframes of a saved motion to the motion sequence in F. Alternatively, a file can also be dragged from the file manager onto the motion sequence F or the sandbox G. The Delete button irreversibly deletes a motion without warning.

## F. Motion Sequence

The motion sequence area with the letter F is most important for motion playback. When pressing the Play button, the motion defined by the set of keyframes currently residing in the motion sequence area is executed by a connected robot and also visualized with the 3D model in the keyframe editor A for offline viewing. While the robot will stop at the last keyframe after a simple Play, the Loop button will periodically execute the same motion over and over again. Please note that the motion speed during the transition from the last keyframe to the first keyframe is defined by the speed parameter of the first keyframe.

The motion sequence area offers basic object handling functionality as commonly known from Windows. Keyframes can be selected with a click. Multiple keyframes can be selected with a rubber band (hold down mouse button and drag) or added to a selection with ctrl-click or shift-click. Keyframes can be generally dragged and dropped with the mouse. When multiple keyframes are selected, all of them will be moved with a single drag and drop operation. Ctrl-C and Ctrl-V can be used to copy and paste selected keyframes. Ctrl-A selects all keyframes in the area. A double click on a keyframe loads it into the keyframe editor for detailed editing. Please note that a loaded keyframe and a selected keyframe are different things distinguishable by distinct border markings. The shortcut keys 1 - 9 load one of the first nine keyframes into the keyframe editor. The + and - keys as well as the mouse wheel together with a pressed Ctrl key zoom the motion sequence area.

The Interpolate button can be used to create an intermediate keyframe linearly interpolated between two selected keyframes. The slider underneath the Interpolate button determines how close the interpolation result is to the either the first or the second selected keyframe. For example, moving the slider entirely to the left would create a copy of the first keyframe. Moving the slider entirely to the right would create a copy of the second keyframe. If the slider is positioned in the center (default), the created keyframe is exactly half way in between the first and second selected keyframes. A double click on the interpolate slider will reset it on the center position. The Delete button will delete all selected keyframes, while the Clear button will delete all keyframes in the area. The Clear button is useful to clear the area before loading a new motion file.

The Filename box and the Save button allow to save motions composed in the motion sequence. If an already existing motion file name is chosen, the old motion will be overwritten with the new one. Otherwise, a new motion file is created. Ctrl-S is a shortcut for pressing the save button. A motion sequence may be permanently saved on the microcontroller for autonomous playback using the Flash button.

## G. Sandbox

The sandbox area denoted as G serves as a temporary buffer to store keyframes. The keyframes residing in the sandbox are not part of the motion sequence. This is also where new keyframes are appended when the save keyframe button is pressed in the keyframe editor or when using the keyframe grabber. The sandbox provides the same object handling functionalities with the same keyboard shortcuts as the motion sequence area.



## List of Keyboard Shortcuts

Ctrl-A, Ctrl-C and Ctrl-V: Selection of all keyframes in a keyframe area (F and G), copy and paste of selected keyframes. Please note that the appropriate keyframe area has to be clicked with the mouse first to set the focus.

Delete: Deletes all selected keyframes in a keyframe area.

+/- : Zooms a keyframe area.

Backspace: Selects and loads the first keyframe in a keyframe area.

Left and right arrow keys (←,→): Selects and loads the keyframe left (right) of the currently loaded one.

Number keys 1 to 9: Selects and loads keyframe number 1 to 9 in the currently focused keyframe area.

0 and ESC: reset all sliders of the Keyframe Editor to 0.

U: Unloads the currently loaded keyframe.

Space and P: Starts the playback of the motion sequence. If pressed during play, it aborts the motion playback.

L: Starts looped motion playback of the current motion sequence.

I: Triggers the initialization process of a connected robot.

Return or Enter: Grabs a keyframe from the robot.

Alt-Enter, Shift-Enter, Alt-Shift-Enter: Toggles full screen mode.

Ctrl-S: Saves the current motion sequence with the file name that is provided in the filename box.

R: Starts the keyframe grabbing operation (Record button).

S: Switches the robot into Stiff mode.

O: Switches the robot off.

Pressing any of the first four joystick buttons grabs a keyframe from the robot.

# Programmer Guide

## PC Software

The software was developed in object oriented C++ using the Qt 4.6 framework. The target operating system is Windows XP. Coding and documentation language is English. Before making actual changes to the code, a solid understanding of C++ and the Qt framework is required. In particular, the signals and slots concept is unique to Qt and heavy use was made of this paradigm throughout the whole software. It is crucial to understand signals and slots to be able to follow data streams in the code and to identify the effects of button clicks.

This guide can offer only a rough overview of the parts the motion editor software is made of and their interconnections. After reading this preliminary documentation it is recommended to inspect the source code of the project. The source files contain rich comments on the purpose of the objects, the implemented methods and information on motion control algorithms that were used.

The graphical user interface itself was created using Qt Designer. It's a wysiwyg editor that allows quick and easy creation and layout of user interfaces. The "drawback" of this technology is that since the GUI code is transparently integrated and compiled into the project, it is very difficult to make changes without Qt Designer later on. The complete elimination of writing actual GUI code by hand is, however, extremely beneficial. The environment used to develop the motion editor project was the Eclipse IDE with the Qt Integration plugin. This plugin integrates Qt Designer into Eclipse and allows a comfortable, seamless workflow. It is recommended to continue development with this IDE, but in principle any toolkit can be used, as long as Qt Designer is available. The style of the GUI is defined in the style sheet "styles.css". If possible, the style sheet should be used to make changes to GUI look and feel. The main advantage is that the software does not need to be recompiled, as the style sheet is applied at loading time.

The number of objects used in the software is manageable, so the source files are kept in a single folder instead of a packaged structure. The project follows the typical object oriented convention that each object is defined in its own source and header pair (.cpp and .h files) and the files are named the same as the object.

The main application object called IgusMotionEditor is the starting point where all other objects are instantiated and the graphical user interface is constructed and launched. Most of the signal and slot connections are made in the constructor of IgusMotionEditor. The handling of the majority of buttons, mouse clicks and keyboard events are handled in IgusMotionEditor as well as internal events and messages such as the successful establishment of the robot connection, connection of a joystick etc.

JoystickControl, KeyframePlayer, KeyframeEditor and RobotInterface are all objects that are involved in the communication with the robot. RobotInterface encapsulates the serial communication with the robot in a separate thread. It receives signals from other objects that contain commanded poses and sends out signals that contain the poses received from the robot by the sensors. The KeyframeEditor is an actual GUI

component that contains the sliders and spin boxes for keyframe editing. When the robot is connected and is in stiff mode, changes of the sliders are sent out as signals to the RobotInterface. The feedback signal from RobotInterface to KeyframeEditor is used for visualization of the currently sensed robot pose with the 3D model. The JoystickControl object is connected to the KeyframeEditor and not directly to RobotInterface. The reason for this is that when the robot is moved by the joystick, the sliders and the 3D model also have to be updated and this is easiest to do when the KeyframeEditor serves as a proxy for the JoystickControl. Essentially, using the joystick is the same as using the mouse to move the sliders of the KeyframeEditor. The KeyframePlayer is responsible for the playback of motions. It continuously sends signals to the RobotInterface during playback and it uses the feedback signal from the RobotInterface to dynamically adapt the joint velocities. The connection from the KeyframePlayer to the KeyframeEditor is used for offline viewing of motions using the 3D simulation model.

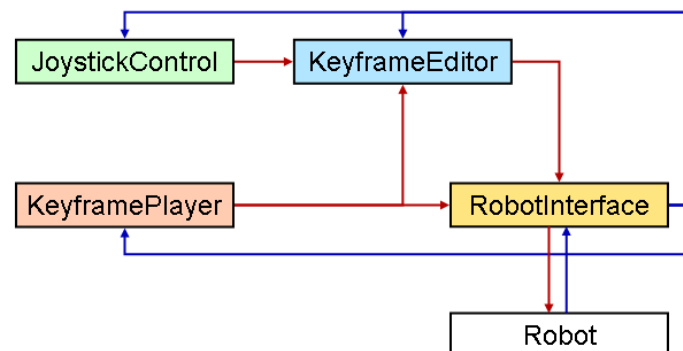


Figure 3: Overview of the components communicating with the robot.

Of course, the motion playback has a higher priority than the joystick or the sliders, so the signals of the other objects are suppressed during playback. Also the selected mode (Stiff or Off) has a strong impact on which object is allowed to send signals to the robot. The signal handling is implemented using Qt's signals and slots concept, which is perfectly suitable for this purpose. The `handleConnections()` method of `IgusMotionEditor` is the central place where the connections between these objects are managed each time a relevant event occurs.

Keyframe and KeyframeArea form a second functional object group that is not involved with robot communication. These objects are only used for user interaction. The Keyframe object consists of the graphical representation of a keyframe on the user interface as well as the actual data that define the pose of the robot. The pose of the robot is a set of joint angles, which are stored in Qt's `QHash<QString, double>` container. The hash structure provides a convenient mapping of joint names to joint angles. These `QHashes` are also used as signals to transfer joint angles and velocities between objects. The KeyframeArea contains Keyframes and supports the drag and drop feature and the handling of mouse and keyboard events related to moving, copying and deleting Keyframes in the KeyframeArea. The motion sequence (F) and the sandbox (G) are both KeyframeArea objects.

The `RobotView3D` object provides the 3D visualization of the robot model. In this particular case, OpenGL was used to describe the visualization.

Figure 4 shows a detailed view of the communication interface at the lower level. The Motion Editor is communicating over a USB connection with the microcontroller board<sup>1</sup> which is tasked with motion playback. The Motion Editor is not needed for playback; autonomous playback may be started using the start button.

The microcontroller communicates with the Nanotec control units over a RS485 bus interface. The number of Nanotec control units is constrained by the microcontroller program. At the moment up to eight Motor drivers can be connected. The joint encoders (sensors) are connected directly to the NanoJ controllers, and are read back using the same RS485 bus.

For playback a motion sequence is transferred to the microcontroller, which gives out position commands to the NanoJ controllers as needed. Inside the NanoJ controllers a custom Java control loop drives to the commanded encoder position.

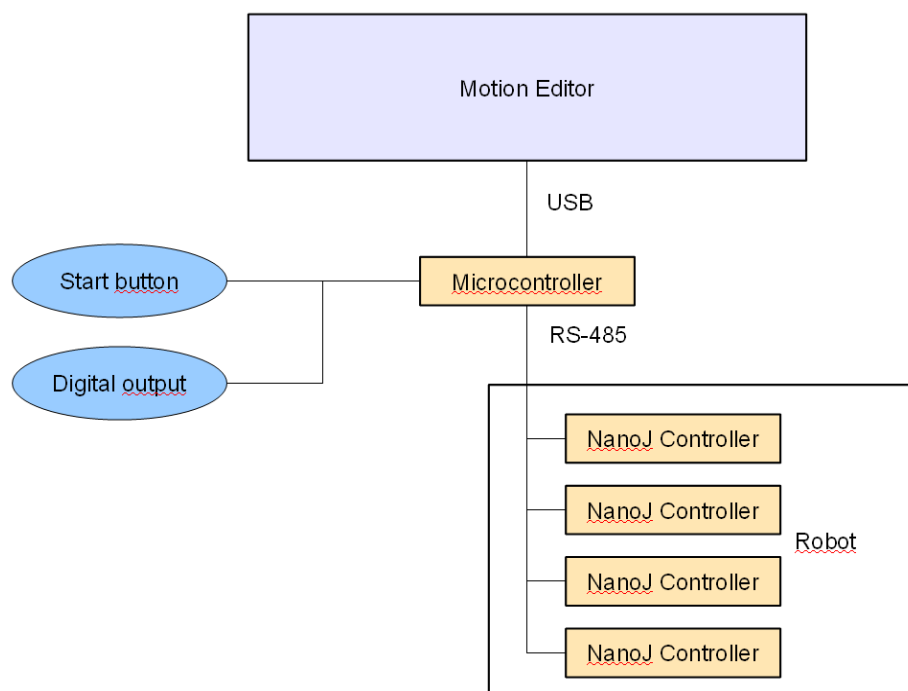


Figure 4: Low level details of the robot communication.

The Java program called “NanoJMotorControl.java” is included in the distribution. It is a fairly simple program that takes care of no more than the initialization of the motor parameters, such as acceleration ramps and driving currents, the initialization procedure of the robot to find the zero position using the HALL sensors and the position control algorithm that tries to correct deviations from the pose commanded by the PC. The Java program can be maintained using the NanoJEasy editor, which is freely available with each Nanotec Motor kit.

<sup>1</sup> Chip45 Crumb2560 available from <http://shop.chip45.com>

## Microcontroller software

The microcontroller software was developed in C++ using avr-gcc/avr-libc. CMake was used as the main build system. A developer might have to change compiler paths and names in CMakeLists.txt to suit his development platform.

### Extended mode

During normal operation the microcontroller acts as a serial bridge between the PC and the Nanotec motor drivers. This enables the user to easily use tools which communicate directly with the motor drivers (e.g. NanoPro). To enable the full feature set, an extended mode has to be entered. This is done by sending a special CMD\_INIT packet. Extended mode packets start with a 0xFF synchronization byte, which cannot appear in normal Nanotec commands (they only use ASCII characters).

In extended mode the serial bridge is disabled, all interaction with the robot takes place using extended mode commands to the microcontroller. A central file is the *protocol.h* header which defines the extended mode protocol. Packets are defined as packed C structures, which are transparently serialized and deserialized.

### Command execution

Extended command execution takes place in *commands.cpp*. This file includes a message packet parser and command execution functions.

### Initialization

Initialization is handled in *main.cpp*. It is automatically triggered when the start button is pressed (see below).

### Motion sequence playback

Motion sequences are played back in *motion.cpp*. The algorithm contains a velocity adaption system that calculates how fast the arm has to move in order to reach future goal positions. This ensures that the robot arm arrives at its target position in time.

In order to autonomously play back sequences, the sequence is loaded from internal EEPROM at startup into a RAM buffer. This RAM buffer is also used to temporarily play back motion sequences from the PC software for testing. If the user is satisfied with the motion, a special CMD\_COMMIT packet commits the motion to EEPROM. All EEPROM handling is done in *mem.cpp*.

During motion playback the system still responds to extended mode commands. This enables position feedback and start/stop control for the PC software.

## Hardware Configuration

### Connecting the Encoder

The connection of the encoder is straight forward, as the labelling of the encoder matches the labelling of the control unit. To connect the Hall-sensor, the analog input of the controller is used (Pin 11). In addition, a ground bridge has to be connected between the ground of the encoder and the sensor input ground (Pin 7), to get a correct input form the Hall-sensor.

### Preparing the controller

Update the controller to the latest firmware.

It is recommended that the Motor-ID is set using the hardware switch to avoid unwanted communication errors. Using NanoPro, the motor has to be configured to Half-Step mode. Also check that the motor and the encoder count in the same direction. If not, it's best to remount the link. If this is not possible, set the Encoder Reverse Flag (line 26 in NanoJMotorControl.java) before compiling and uploading. Then, using NanoPro, set the controller to automatically load the Java program.

### Compiling and uploading NanoJMotorControl

Open NanoJEasy and load NanoJMotorControl.java. Check the values set in the first 50 lines. Those are automatically set with every boot of the controller. Line 20-24 set the Encoder-resolution according to the Motor-ID. They are calculated as follows:

round up (  $2 \times \text{encoder-resolution} \times \text{gear-reduction}$  )

Example: motor 1 and 3 are set to  $( 2 \times 4960 \times ( 1/15 ) ) = 661.33 \rightarrow 662$

Setting the driving currents for the motors requires special attention, as different motor currents are used throughout the initialization and the process to find the Hall-sensor. Line 43 sets the current for the controller, 50 at the moment. This means 50% just like in the NanoPro program. If another motor is used, try this value with the NanoPro program and adjust it accordingly. The same current is also set in line 126, where the Hall-finding procedure resets it after completion. The Hall-finding procedure temporarily uses a lower current in line 61 to prevent driving with too much force into a joint limit. This temporary value should also be tested with NanoPro. It should be just high enough so that the motor can move the joint. Line 44 sets the current when the motor is not driving. This should be set to a value where it is not easy to move the link by hand when holding a position.

Now, after checking these parameters, compile the program.

If any parameters are altered, check the motor number before uploading, so that you upload the compiled program with the right parameters to the right motor!

## Microcontroller connections

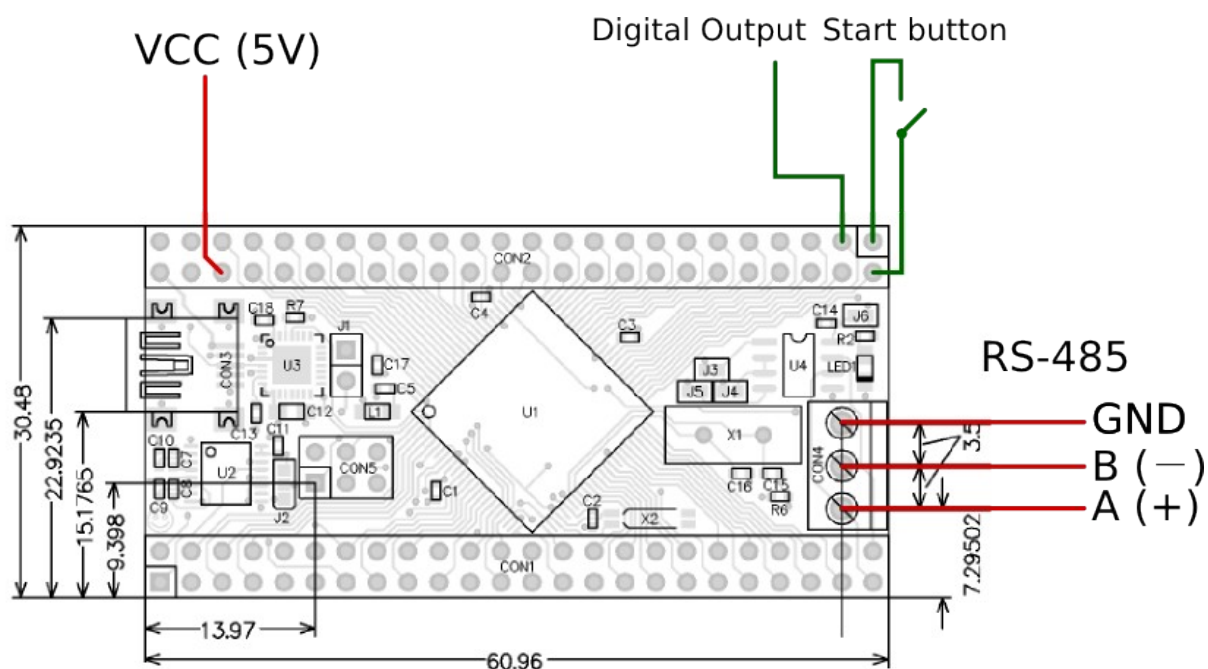


Figure 5: Microcontroller connections

Fig. 5 shows how the microcontroller board should be connected. An operating voltage of 5V has to be provided and connected to the appropriate pins. The digital output is a TTL level active-high push/pull transistor output which can source and sink up to 40mA. For more details, see the Atmel ATmega2560 datasheet.

To enable the RS485 interface a few solder jumpers have to be set on the board. Please refer to the board documentation for details.

## Start button semantics

On a start button press the microcontroller goes through three phases:

1. **Initialization:** The joints move and search for the encoder index position. This phase is skipped if the joints are already initialized.
2. **Move to start position:** The robot arm moves to the first keyframe.
3. **Motion playback:** The motion sequence is played. If the start button is still pressed at the end of the sequence, playback is looped smoothly.

A short button press executes only one phase, so three presses are usually required before playback starts. If the button is held down, all phases are executed automatically.

## Microcontroller firmware

The microcontroller needs firmware to operate. A small GUI application called "flashtool" (see Fig. 6) is included in the software distribution which guides the user through the initial bootloader and firmware flashing or a firmware update. An AVR programmer is required for the installation of the bootloader. Depending on the programmer, the microcontroller may need to be externally powered during programming.

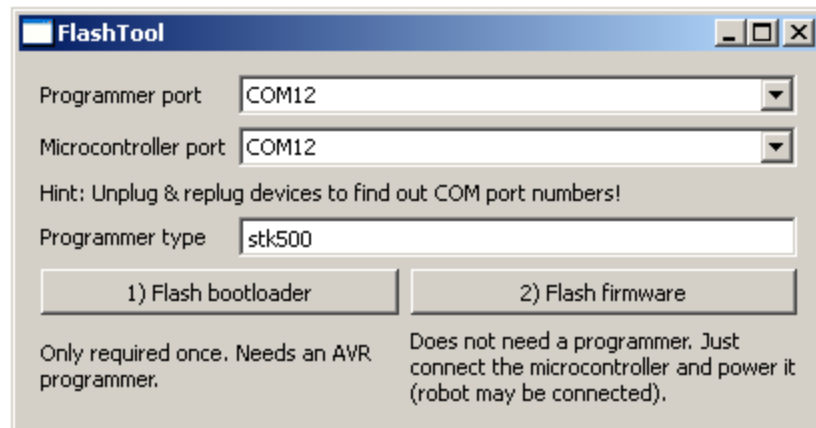


Figure 6: Microcontroller flash tool