T-Bone Software Documentation

The T-Bone Cape

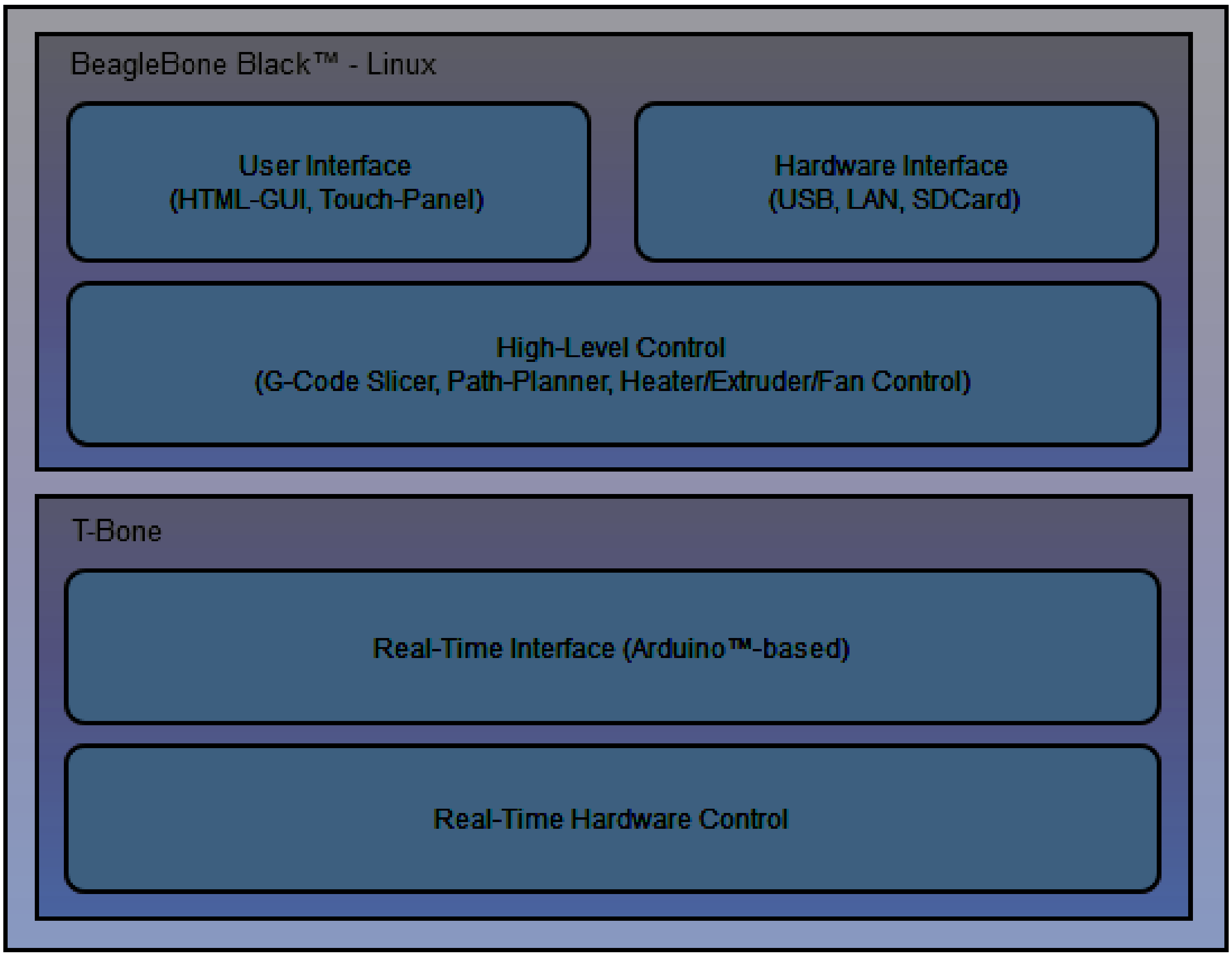
The T-bone is a cape for the BeagleBone Black (tone.cc), dedicated for motion control application. These are 3D printers, laser cutters, milling machines, and other applications using stepper motors. The T-bone will come preprogrammed for Reprap Mendel 3D Printer (http://www.reprap.org/wiki/Mendel). Best kown in the Prusa Medel (http://reprap.org/wiki/Prusa_Mendel_(iteration_2)) or Prusa i3 (http://reprap.org/wiki/Prusa_i3) variant.

Introduction

T bone blends motion control, realtime motion execution and 3D printing software in a n easy to digest package. User interface, configuration, and path planner are running on the BeagleBone Black. This makes it easy to adapt. Real-time communication with the BeagleBone is handled by a small microcontroller, placed on the T-bone. The microcontroller is fully compatible with the Arduino toolchain. Complex acceleration and velocity calculations for the stepper motors are done by dedicated motion controllers. These are dedicated hardware components, developed to get the maximum performance out of a given stepper motor without putting any workload to the host system.

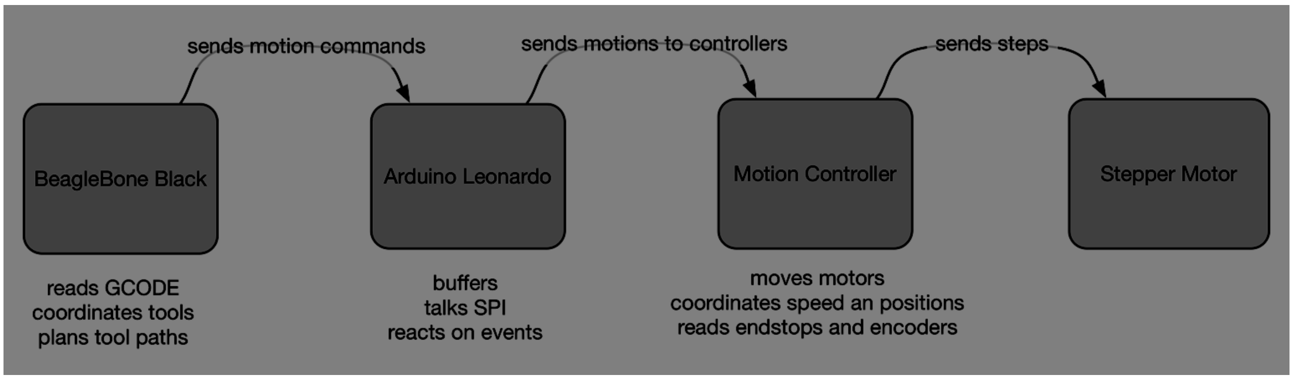
Base Architecture

The idea of the T-Bone software architecture is to use to reduce the real time requirements for the motion control part and implement that in a high level language (here python). To achieve this the motions of the squirrel motion controller are used as timings.


BeagleBone Black™ - Linux
User Interface (HTML-GUI, Touch-Panel)
Hardware Interface (USB, LAN, SDCard)
High-Level Control (G-Code Slicer, Path-Planner, Heater/Extruder/Fan Control)
T-Bone
Real-Time Interface (Arduino™-based)
Real-Time Hardware Control

The BeagleBone is used as User Interface provider and high level path and motion planner, while the Arduino Leonardo is used as Real Time Interface to translate between the high level concepts of the T-Bone python part and the hardware.

The architecture is a pipe and filter architecture, containing several buffers which are kept at a certain level to make the different software parts independent from each other in timing. By thish slight timing variations can be easily compensated.



The Python Code on the Beagle Bone reads the GCODE, interprets and recalculates the movement (e.g. from inches to mm). The BeagleBone also hosts the web interface and contains the server part. During GCODE interpretation it also applies motion planning over several segments of a motion.

The Arduino Leonardo keeps a planned motion buffer and coordinates it's execution in the motion controllers. It also hides details of the squirrel programming from the python part. The Arduino Leonardo is connected by serial connection to the python part and by SPI to the squirrel motion controllers.

The motion controllers itself know just two motions: the currently executed motion and the next motion to execute and coordinate the stepper motor.

Software Modules of the Python Server

The T-Bone Server

The module 't_bone_server.py' – especially contain the command line interface to start the server (in the main routine) and the service endpoints for the web interface (the controllers in a model-view-controller architecture). The web interface itself is rendered with the HTML templates in 't_bone_server/src/t_bone/templates' containing a base template (base.html) with all the header and footer information which are the same for all pages. The pages itself are realized as sub templates.

The web server itself does not have any logic to interpret GCODE or configure the 3D printer itself.

The 3D Printer

The module 'printer.py' contain the high level abstraction of the reprap 3D Printer. It knows is axis, it knows how to read the configuration. It knows that the axis correspond to motors. It translates between real world coordinates in millimetres and stepper motor coordinates in steps. It knows that a path to move consist of several parts and that you do not have to stop at the end of each segment.

The 'Printer' is it's own thread which coordinates the movements itself. The configuration is done by handing a dictionary (or JSON file) to the configure() routine. This routine interprets the information and sets the member variables of the printer and axis accordingly.

A print is started with 'start_print()', the several movements can be added with 'move_to()' and in the end the print is finished with 'finish_print()'. The print buffer is split in two separate buffers: The planning buffer in the class 'PrintQueue' (as member variable _print_queue). This planning buffer has a fixed length and is used for motion and path planning. If an move is added to the print queue with 'add_movement()' four steps are performed:

- the maximum achievable speed for the move is calculated. Depending on the speed and direction of the previous move and the motion parameter configuration the print head can be further accelerated or must stop and accelerate from 0.
- It can be decided if there is a stop for an axis in the previous move or the motor can just move on. After this the configuration of the previous move is finished and the previous move can be added to the planning list in 'planning_list'.
- If the planning list is long enough the first move is removed and pushed to the execution list in _push_from_planning_to_execution().
- After this all moves in the planning list are recalculated in _recalculate_move_speeds() to check if we work from the back of the planning list to the front of the planning list we can brake fast enough to achieve the target speeds of each move. If the achievable speed is higher than the highest possible brake speed it must be reduced.

The execution list in 'queue' is a simple Queue with a maximum length which is used by two separate Threads:

- The printer pushed executable movements to the queue
- The print queue reads executable moves and executes it on the machine (see next paragraph) if there is enough space in the Arduino Leonardo internal execution queue

The GCODE Interpreter

For simplicity reasons the GCODE interpreter itself is kept in the 'gcode_interpreter.py' module. It contains logic to read from machine specific GCODE to the API of the 3D Printer. There is no direct connection between the 3D Printer and the GCODE to allow easy adaption and change of GCODE interpretation.

The low level Machine Interface

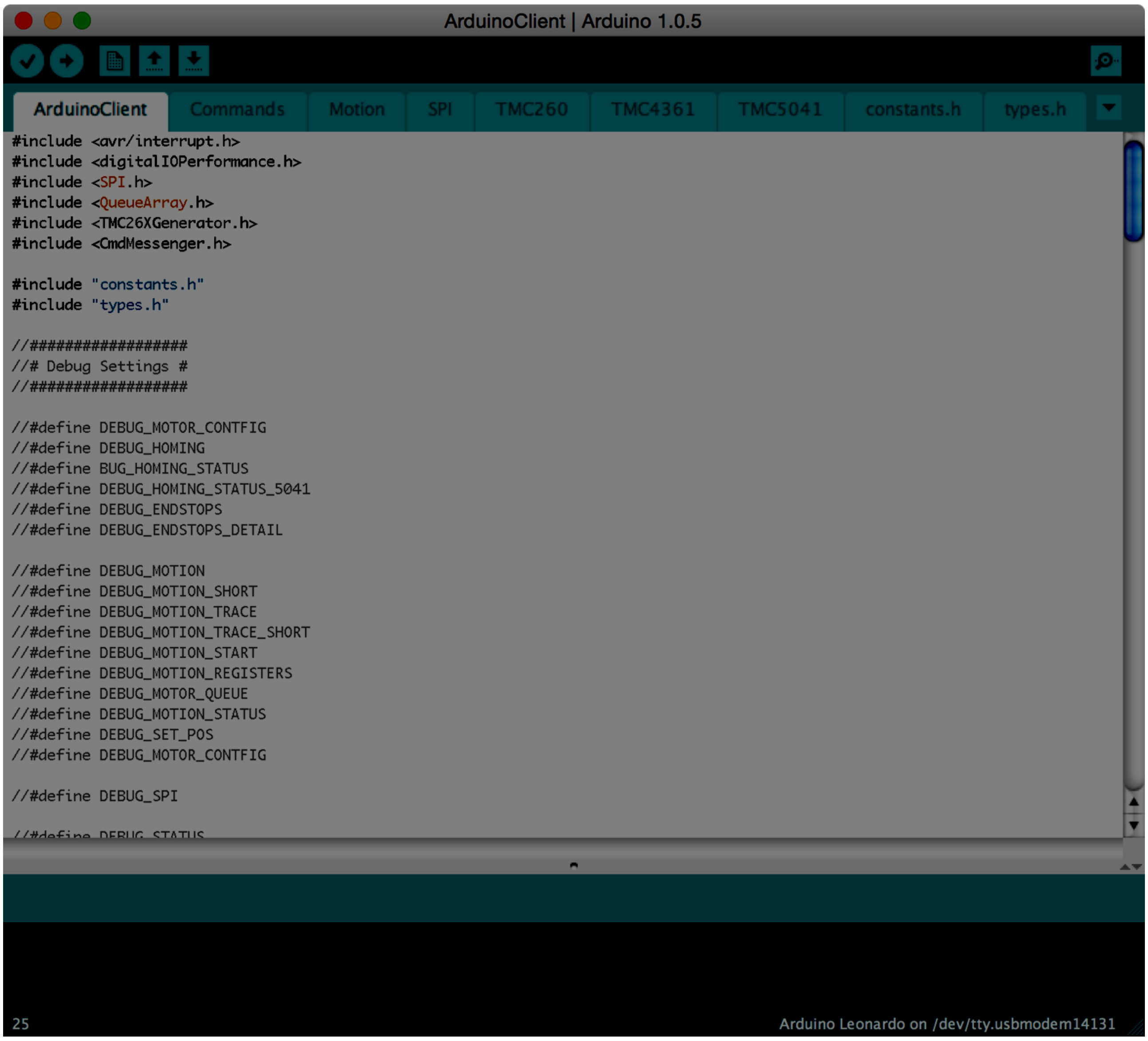
The module 'machine.py' contains the serial communication interface to the Arduino Leonardo. It is a separate Thread which ends requests, receives and decodes answers and is also notified by heartbeat pings from the Arduino Leonardo to update status information. It translates between the object language of the printer and the technical language of the serial communication.

The Heater

The module 'heater.py' contains the logic to keep a heater at a constant temperature. A heater consist of a thermistor and a PWM output. The heating logic can either be a PID controller (for the Hotend) or a simple two point controller (for the heated bed). The Thermistor logic is separated in the 'thermistor.py', 'replicpe_thermistors.py' and 'ramps_thermistors.py' packages.

Software Modules of the RealTime Part in the Arduino Leonardo

Due to the limitation in space and programmatic expressions of the Arduino Language (even though it is mostly C++ minus some possibilities) the structure of the Arduino Part is not as differentiated as the python part. The different functions are separated into different files which are represented as different Tabs in the Arduino IDE.



```
#include <avr/interrupt.h>
#include <digitalIOPerformance.h>
#include <SPI.h>
#include <QueueArray.h>
#include <TMC26XGenerator.h>
#include <CmdMessenger.h>

#include "constants.h"
#include "types.h"

//##################
//# Debug Settings #
//##################

//#define DEBUG_MOTOR_CONFIG
//#define DEBUG_HOMING
//#define BUG_HOMING_STATUS
//#define DEBUG_HOMING_STATUS_5041
//#define DEBUG_ENDSTOPS
//#define DEBUG_ENDSTOPS_DETAIL

//#define DEBUG_MOTION
//#define DEBUG_MOTION_SHORT
//#define DEBUG_MOTION_TRACE
//#define DEBUG_MOTION_TRACE_SHORT
//#define DEBUG_MOTION_START
//#define DEBUG_MOTION_REGISTERS
//#define DEBUG_MOTOR_QUEUE
//#define DEBUG_MOTION_STATUS
//#define DEBUG_SET_POS
//#define DEBUG_MOTOR_CONTFIG

//#define DEBUG_SPI

//#define DEBUG_STATUS
```

25 Arduino Leonardo on /dev/tty.usbmodem14131

The Arduino Client itself

The module ArduinoClient contains the basic configuration of the Arduino Client the basic setup and loop routines to keep the program going and the include definitions to keep everything together. No special functions are implemented in this module.

The Commands Module

The Commands Module contains the interpretation and answering of the serial communication commands. It contains no real logic just the commands, it's numbers and parameters, the calls of the corresponding functions and the replies.

The Motion Module

The Motion module contain the main motion buffer (moveQueue) ,the information if currently a coordinated motion is executed (current_motion_state). If it is so it knows is a move is currently executed (move_executing) or the next move is prepare in the squirrel shadow registers (next_move_prepared).

The move queue is a ring buffer which is filled from the serial interface and emptied by executing moves. If the next move is not prepared a move is read from the motion queue and written to the squirrel shadow registers. If an move is executed it tracks if all motors of the move (motor_status) reached it target (target_motor_status). If all motors reached the target a new start signal is send.

The SPI Module

The SPI Module contain the functions to read and set registers in the squirrel and TMC5041.

The TMC260 Module

The TMC260 Module contain some functions to write register values to the TMC260 connected to the squirrel motion controllers and basically initialize the TMC260 after booting. The configuration of the TMC260 itself is done with the TMC26XGenerator – which is derived from the TMC26XDriver.

The TMC4361 Module

The TMC4361 Module contain the setup code to correctly set the pins connected to the TMC4361 and and the functions to translate between the high level concepts and the TMC4361 registers.

The TMC5041 Module

The TMC5041 Module contain the setup code to initialize the TMC5041 after boot and the functions to translate between high level concepts and the TMC5041 register

The constants

The file constants.h contains all constant values and conversions used in other files. They are collected here to get them out of the way for other modules.

The types

The file types.h contain the typedef definitions to have them in one convenient place.

The Serial Communication Interface

As Serial Communication Interface CmdMessenger was choosen. It had the advantage of being human readable and writeable and implemted. An message or answer always consists of a number of comma separated values:

<command number>, <option 1>, <option 2> and so on. A command is ended by ';'

A example command could be 1,12.0,2,"test"; and be answered by 0,0;

Commands can contain options and parameters for integer values, floating point values and text. On Arduino integers and long are separated since on Arduino integers are only 16 bit wide.

Action Commands have positive command numbers. Answers have negative command numbers. 0 means OK.

Command 1: Set Motor Current

This command set the motor current for any axis. The parameters are

- The motor 1-5
- The current in mA

The current can only be set for the squirrel motors currently.

It is replied with OK or an error.

Command 2: Configure Encoder

This command is used to configure the encoder of an TMC4361 motor. It contains the following options:

- The motor 1-5
- -1 for disabling the encoder, 1 for enabling the encoder
- the steps per revolution for the motor
- the microsteps per step for the motor
- the increments per revolution for the encoder
- 1 if the encoder is differential, -1 if not

The last 4 options can be omitted if an encoder is disabled

Command 3: Configure Endstops

This command is used to configure the endstops of a motor. It contains the following options:

- The motor 1-5
- 1 for right and -1 for left endstop position
- 0 for real and 1 for virtual endstop
- for virtual enstops
  - the endstop position
- for real endstops
  - -1 for an active negative and 1 for an active positive endstop

Command 4: Invert Motor

This command inverst the direction of a motor movement. The parameters are

– The motor 1-5
– 1 for a non inverted motor, -1 for an inverted motor

If the second parameter is 0 the reply is OK and if the motor is inverted (e.g. 4,1,-1;)

Command 9: Initialize Motion

This command initializes all chips, resets the TMC4361 and flushes the movement queue.

Command 10: Add Move

The add move command adds a move of several motors to the motion queue. For each motor the following information is given:

– The motor 1-5
– The target of the movement as step position
– The movement type 's' for a movement which stops at the given endpoint or 'w' for a movement which is continued after the current way point.
– The maximum speed of the movement in steps/s
– The maximum acceleration of the movement in steps/s^2
– The bow or jerk of the movement in steps/s^3

Command 11: Movement information and finish movement

The movement command can be used to read out if the machine is currently executing a movment or to indicate the machine to finish a movement and empty the movement buffer.

It consist of these parameters:

– 0 for get movement information or a negative number to indicate to finish the movement

Command 12: Home Axis

This command can be used to drive a motor to the endstop to achieve it's actual physical zero position. The homing is achieved by first driving with the homing fast speed to the endstop, go back by the given retraction and slowly driving back into the endstop with the slow homing speed to achieve an very precise hit.

It consists of the following options:

– The motor 1-5
– the homing timeout
– the fast homing speed
– the precision homing speed
– the retraction after the motor first hits the endstop
– the acceleration of the homing movement
– the bow or jerk of the homing movement

Command 13: Set Position

The set position command can be used to set the internal step position of a motor. It consists of the following options:

- The motor 1-5
- The new position to set.

These positions are added to the move queue to move the motor to a position and set it there.

Command 30: Read Position

This command can be used to read out the current motor position. It is not queued and answered directly. It contains the following options:

- The motor 1-5

The Answer consist of the following information:

- The current position

Command 31: Queue Status Readout

This command is answered with a number of queue status options:

- The current length of the execution queue
- The maximum length of the execution queue

Command 32: Read Status

This command reads the motor status. It contains the following options:

- The motor 1-5

This command is answered with some status information for the given motor.

For TMC4361 motors:

- the status flags register
- the x actual register
- if the left endstop is active
- if the right endstop is active
- the the encoder position register

For TMC5041 motors:

- the x actual position

Command 41: Analog Reading

This command gives back the analog reading of one of the Arduino Analog ports. It can be used to read out the current on the hotend or the heated bed.