Text to Motion Database

System Architecture

Brendan Duke
Andrew Kohnen
Udip Patel
David Pitkanen
Jordan Viveiros

April 9, 2017

Contents

Revision History

Date	Version	Notes
January 5, 2017	0.0	File created
April 9, 2017	1.0	Final documentation revision

1 Overview

This document contains a top-down description of the system architecture of the Text-to-Motion Database/Software Suite (hereafter referred to as Text-to-Motion). Design decisions are described here at a high level, along with a module/component decomposition. Challenges from previous deliverables are discussed, and those challenges' solutions described. Explicit traceability between modules and requirements is given.

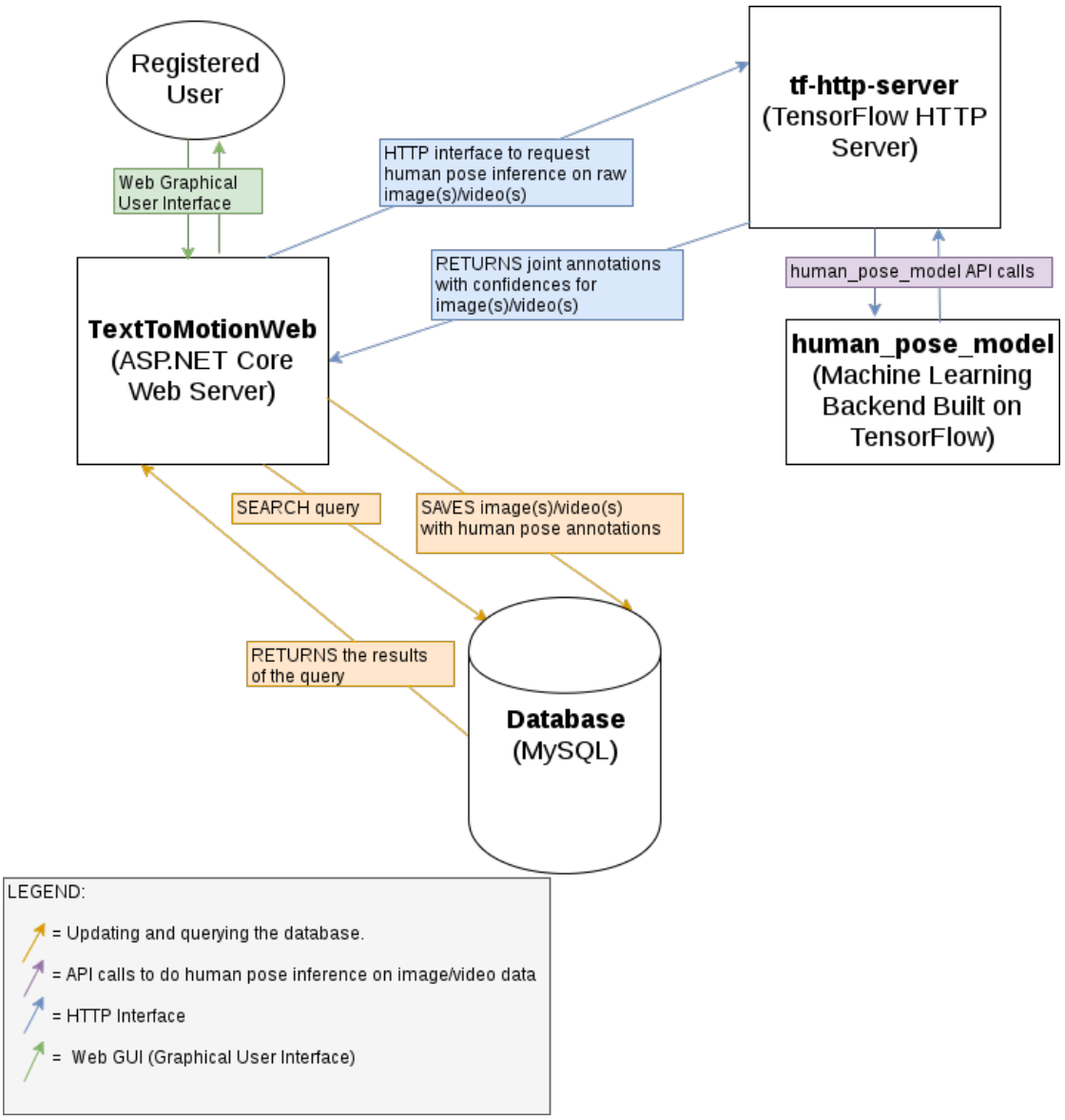
A detailed description of the components' specification and implementation is given in the "Detailed Design" document.

2 Component Decomposition

A high level, block-diagram style overview of the Text-to-Motion software architecture is shown in Figure 2.1. The modules are broken into four major components: the `TextToMotionWeb` ASP.NET Core web server, the MySQL database for storing images, videos and human pose annotations, the `tf-http-server` TensorFlow HTTP server, which serves human pose inference, and the `human_pose_model` software module, which provides an entire solution for training and evaluating a human pose estimation model, and running inference with that model.

In the following sections, an overview of the top-down functionality of each module shown in Figure 2.1 is given.

Figure 2.1: A top level view of the architecture of the Text-to-Motion software suite.



2.1 TextToMotion Web Server

The `TextToMotion` web server, hosted at `https://brendanduke.ca`, serves as the GUI (Graphical User Interface) to the entire project. The TextToMotion web server is also the component that connects the web interface, human pose inference model, and database together with controlling logic for user requests.

The TextToMotion web server follows the MVC (Model View Controller) architecture in the manner encouraged by the ASP.NET Core framework. In the case of the

Text-to-Motion project, the Model defines a model for video, human pose annotation, and associated rich text annotation data, which is stored in the database. The Views define the appearance of the different web pages. The Controller directs the flow of requests, which originate from the web interface, to the human pose inference or database components.

The TextToMotion web server provides a web interface through which users can make requests to search, upload and retrieve human pose video data. Requests made to upload video data will cause the TextToMotion web server to communicate over HTTP protocol to the `tf-http-server` module, and thereby retrieve human pose annotations for the uploaded video data. The TextToMotion web server then converts the retrieved human pose annotations into HDF5 format and stores human pose data along with video and any uploaded text annotations into the database.

A search query interface is also provided by the TextToMotion web server, whereby users can use a keyword search to retrieve videos with their respective human pose and text annotations. The TextToMotion web server retrieves these data through a query to the database.

2.2 tf-http-server Human Pose Inference HTTP Server

The `tf-http-server` module provides an HTTP interface through which human pose inference can be done on images or video. The `tf-http-server` module is stateless in the sense that it takes input and returns the output corresponding to that input, without storing or depending on any current state of the `tf-http-server` module.

The `tf-http-server` module is written in Python, and interfaces directly with both TensorFlow and the `human_pose_model` module.

It is possible to run the `tf-http-server` software on the same physical server as the `TextToMotionWeb`, or, as is done in the current TextToMotion deployment, these two software modules can be run on distinct physical servers. In the current deployment, `TextToMotionWeb` is run on a VM (Virtual Machine) rented from Digital Ocean, while the `tf-http-server` is run from a machine with a GPU in the Guelph lab.

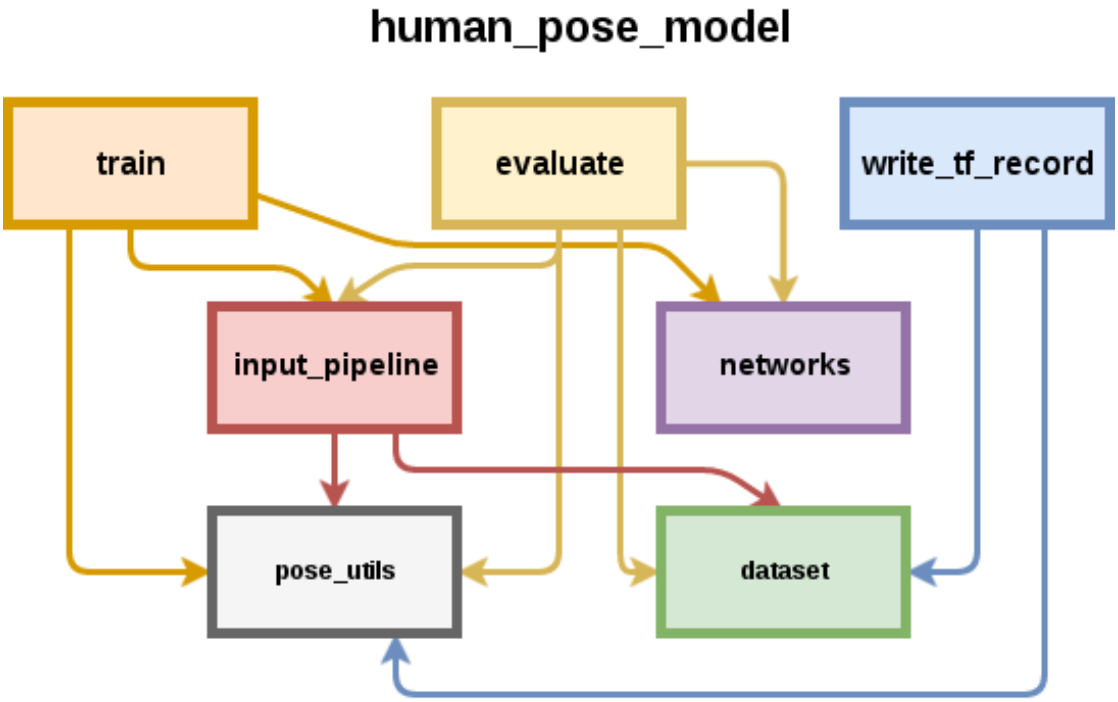
2.3 human_pose_model Human Pose Model Training and Inference Software Suite

The `human_pose_model` module provides a completely standalone software suite for the training, evaluation and deployment of human pose inference models.

A uses hierarchy for the `human_pose_model` module and its sub-components is given in Figure 2.2. In Figure 2.2, the arrows from one module to another signify that the module at the tail end of the arrow uses, i.e. imports, the module that the head of the arrow is pointing to. The uses graph is a hierarchy because there are no cycles in the arrows formed by the uses relationships.

In the following sub-sections the functional uses of each of the `human_pose_model` sub-components is given.

Figure 2.2: human_pose_model uses hierarchy.



2.3.1 train

The `train` submodule trains a human pose estimation network to detect and/or regress binary maps and/or confidence maps of joint co-ordinates (`NUM_JOINTS` sets of (x, y) co-ordinates). A configuration file is expected, which specifies training parameters, which network to train, as well as the location and dimensions (width and height) of training and validation data.

The `train` sub-module calls the `input_pipeline` module in order to setup a pipeline to retrieve and preprocess training data. The `networks` module provides an interface to access by name the network chosen in the configuration file, and thereby obtain TensorFlow gradient ops for back-propagation, as well as to do inference with the chosen network.

2.3.2 evaluate

The `evaluate` sub-module computes an evaluation metric, namely the PCKh metric (an accuracy metric for human joint positions), using the latest checkpoint (saved automatically by the TensorFlow software) in a specified directory.

The `evaluate` sub-module uses `input_pipeline` to create a data input pipeline for evaluation of the validation data, the `networks` sub-module to do inference with the given network and the `dataset` sub-module to gain access to the `NUM_JOINTS` parameter

for the dataset.

2.3.3 write_tf_record

`write_tf_record` is a sub-module that creates set of TFRecords, which are a standard format for TensorFlow data. The TFRecords consist of ground truth human pose annotations along with their respective images.

The `write_tf_record` module uses the `dataset` module to get information about the format of the human pose dataset.

2.3.4 input_pipeline

Sets up an input pipeline that reads example protobufs from all TFRecord files, decodes and preprocesses the images. The input pipeline is setup with a sequence of queues so that preprocessing and file-reading can be parallelized across many hardware threads.

2.3.5 networks

The `networks` sub-module contains interfaces to obtain networks by name, e.g. the "two VGG-16s cascade" network, to do inference with those networks, as well as to compute different loss functions. The network and loss function are independently configurable via the configuration file.

2.3.6 dataset

The `dataset` sub-module is a module for reading the MPII Human Pose Dataset.

The MPII dataset is a set of 25 000 images, containing annotations for 40 000 people in total. The images are colour and of various dimensions (e.g. 1280x720, 640x480 etc.).

The ADTs (Abstract Data Types) for a `Person` and a `Joint`, including the `NUM_JOINTS` parameter, are provided by the `dataset` sub-module.

2.3.7 pose_utils

The `pose_utils` sub-module provides a set of commonly used utility functions to most other modules. The utility functions provided by `pose_utils` including image utilities used for drawing confidence maps on images, profiling modules (`timethis.py`), and convenience functions for converting between sparse and dense Tensors.

3 Important Design Decisions and Reasoning

3.1 Reason for Component Decomposition

The reason for the given module breakdown is to separate the responsibility of different modules amongst the clearly distinct tasks performed by the overall Text-to-Motion solution. The advantage of the separation of responsibility is that modules can be more easily unit tested, can be re-used as part of other software projects, and can be replaced with other modules with similar functionality.

Specifically, the `TextToMotion` ASP.NET Core web server component has been kept distinct from the data storage component and the human pose inference component. Throughout development this has enabled the complete switch of data storage module (from SQLite to MySQL) as well as the human pose inference module (from a C++ interface with a Caffe deep learning backend, to an HTTP interface that uses TensorFlow as the deep learning backend).

The Text-to-Motion software architecture's specific module breakdown also allows for the flexibility of re-using the modules in different projects. For example, any project can re-use the `human_pose_model` module to train their own human pose inference model on their own dataset. Furthermore, any software capable of interfacing with HTTP will be able to drop the `tf-http-server` module into their project and use it without any changes.

3.2 Reason for HTTP Interface to Human Pose Estimation Backend

As mentioned above, a major motivation for using an HTTP interface to the human pose inference module is to make the module flexible enough to be used in any software project, as required by Requirement 38, which mandates the software's usability from any major operating system.

Using the HTTP interface also allows for flexibility in the physical setup of the servers. Current deployment uses a VM in the cloud as the `TextToMotion` server, which is useful for reducing cost, as well as ease of remote management of the server. However, the VM does not have a virtualized GPU, which is needed to meet the speed and latency requirement of Requirement 24. To offload inference computations to a GPU, the VM maintains an SSH tunnel to a machine in the Guelph lab that does have a GPU, and is running the `tf-http-server` to serve human pose inference requests over the network.

3.3 Design Decisions Behind human_pose_model Module

The `human_pose_model` module was written in Python to satisfy the Solution Constraint Requirement 1. By writing an end-to-end software suite for training a human pose estimation model, the constraint that modern deep learning methods are used can be ensured by allowing the flexibility of changing and re-training the model as research progresses.

An explicit constraint, Requirement 3, requiring the Text-to-Motion Software Suite to include a software module for training a state-of-the-art human pose estimation model from scratch was included once it became clear that existing off-the-shelf software would not be sufficiently accurate to satisfy Requirement 9, which mandates spatial and temporal accuracy of the human pose inference model. The `human_pose_model` module satisfies Requirement 3.

Solution Constraint Requirement 5, which says that software modules should be accessible from Python, is satified by writing `human_pose_model` entirely in Python.

3.4 Challenges from Previous Deliverables, and Solutions to Said Challenges

One challenge from previous deliverables was that, based on the performance of the deep learning model that was used in the prototype, our existing human pose estimation model was not accurate enough to meet Requirement 9. The inaccuracy of the off-the-shelf model used in the Text-to-Motion prototype was clear qualitatively, as unoccluded joints were frequently predicted completely wrong.

As a long-term solution to the challenge of the lack of off-the-shelf human pose estimation software, the `human_pose_model` software component was developed. Rather than continue to search for different off-the-shelf software, a complete solution for training one's own human pose estimation model has been provided. Providing software to train one's own model has the added benefit of meeting Requirement 3, which is a requirement that was brought up in the middle of project development based on our users' desire to participate in current human pose estimation research.

Another challenge from previous deliverables was that the human pose inference did not run fast enough to meet Requirement 24 when run on a CPU, and the development team does not have the funds available to rent GPU hardware for use with the web server. As a solution for the computation problem, an interface to communicate with the human pose inference backend by HTTP protocol was designed, and the `tf-http-server` module was developed. The `tf-http-server` module runs on a machine with a GPU provided by Dr. Taylor in the Guelph lab, and this machine is accessed via SSH tunnel (port forwarding) by the web server with a public IP and domain (`https://brendanduke.ca`).

4 Anticipated and Unlikely Changes

As the Text to Motion Database continues to grow and be developed there are inevitably going to be changes in the design and structure of the overall project. The next sections aim to outline the aspects of the project that may be prone to change versus the aspects that are core to the functionality of the database.

4.1 Anticipated Changes

- **User Interface Design:** The user interface will likely be changed throughout the development process due to feedback from users and supervisors. The changes should improve the user experience and add functionality when possible in order to promote the usability of the database through the web interface.
- **Deep Learning Network:** The deep learning network will see improvement through the accuracy of the pose estimation performed on uploaded images or video. This may occur by re-training the existing network by the development team, or more state of the art methods and algorithms can be applied.

4.2 Unlikely Changes

- **Image/Video Upload:** At the core of the McMaster Text to Motion Database the ability to upload video and images is one of the primary functions and will remain an option throughout the project's development.
- **Database Search:** The ability to search through the database is something that the Text to Motion Database should always provide and will evolve when the full text search is implemented.