Beats by Jetson

Jetson TX2 is a powerful platform that excels in on-board computing and real-time inferring. Although there are many applications of Computer Vision and Robotics on Jetson platforms exist, there is a visible lack of audio processing projects that utilizes these systems. The Jetson boards do lack the sound cards but existence of real-time audio processing modules such as Magenta, Aubio, IBB, and madmom encourage using USB devices or the I2S interface for projects focusing on sounds.

With this in mind, we invite you to imagine you can have a smart percussion bot, that can play along with you, attend jam sessions and one day replace one of your band members. Extracting beats from an audio source would be its first task, and it would utilize trained models or well tuned classifiers. But that wouldn't be up to the task as trying to play catch up only using detections would fail just with the addition of the actuation lag. So, such an automated player would need to predict beats and patterns as well as having a good musical ear to jam with his human (or bot) friends!

In this report we are presenting a prototype percussion player that detects on-set beats and predicts next ones in order to play along sound input in real time as it operates a bass drum pedal with a stepper motor.

System

The software consists of three elements: a beat detector, a next-beat/rhythm detector and the pedal driver. All nodes utilize rospy.Time() for synchronization, timestamp calculation and loops. Our custom ROS messages are included in the beat_msgs package.

Beat_detector.py implements the audio interface of our software. It loads an audio file or a line in channel using PyAudio and executes a callback to process a window of samples. This is where *aubio* beat module is used to publish detected beats.

Master_of_beats package contains the future beats from the observations. It can also predict from a given text file of beat timestamps or simply relay incoming beats to the player. This node imports our models in rhythm.py. Each incoming beats updates our observations and trigger a prediction depending on the model. The main loop acts as a timer to execute player's commands or update the time label as needed.

There is also a simpler script that publishes beat messages on key press, which is useful for testing and more importantly, **learning-from-demonstration** applications.
Driver (*drummer/whiplash.py*) controls a stepper motor by sending one step command in each loop. Incoming Booleans that represent predicted beats on "/thump" topic invoke the callback to

set the motor direction towards the drum and increment the count of unplayed beats. Pedal always moves towards the rest position slowly or to the drum aggressively, except if a bea issued when the pedal head is too close to the drum, it will only back up a little only to hit the drum quickly.

Future Development

The layout of the software enables us to implement much more sophisticated sound detection and pattern prediction algorithms. Just by adding a classifier or a trained model to te callback function of *beat_detector* more information can be extracted from sampled buffers and published to the listening modes. Our goals on this end include extending these to labeling different percussion hits, detecting tempo and bars to build detailed percussion tracks and extracting chords in order to predict chord progressions (with variations).

Currently the hardware lag is compensated by a predefined parameter. This can easily be automate by sending a service call or message to the *beat_player* (predictor) on the completion of motor's forward steps to compare timestamps. More service calls can be added to switch between training, model building and confidently playing.

The hardware can reach faster execution by using a linear actuator with spring load and rotary trigger action.