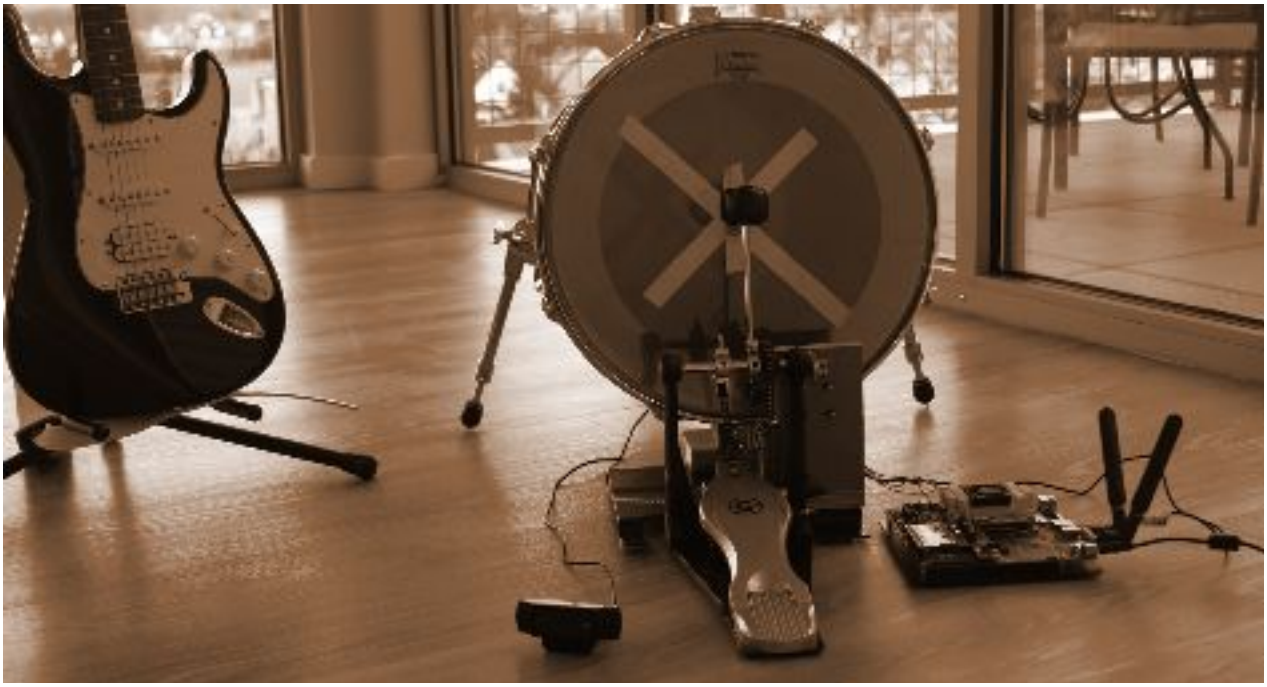
# Beats by Jetson

Joseph McMahon Sarmad Sharif Ozan Akyildiz



## Introduction

Jetson TX2 is a powerful platform that excels in onboard computing and real-time inferring. Although there are many applications of Computer Vision and Robotics on Jetson platforms, there are not many audio processing projects. The Jetson boards do lack the sound cards but the 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 of 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. We utilize the processing speed of Jetson TX2 to achieve higher precision on timestamp extraction and timed-event executions on high quality (35-44.1KHz) audio input

# System

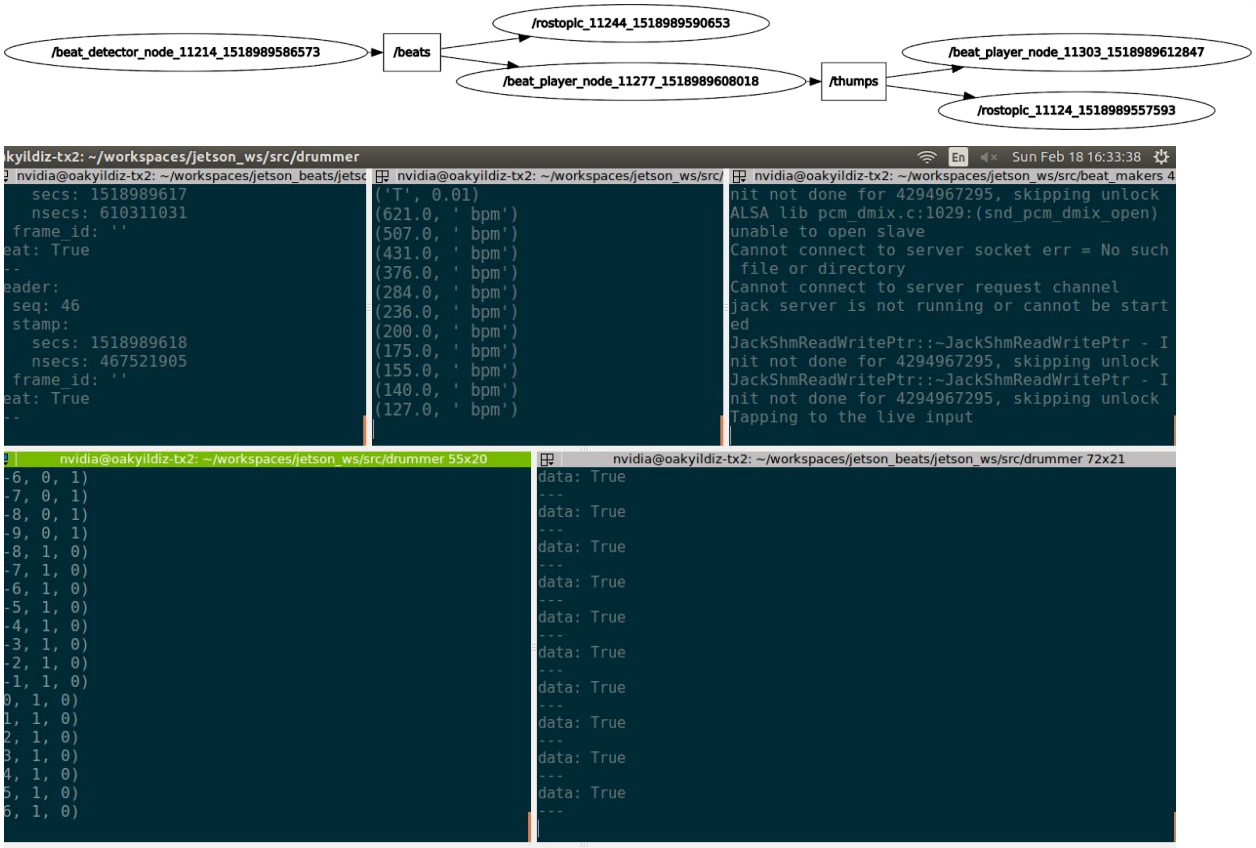



Figure 2a: rqt_graph overlay of the ROS nodes  2b:Console output. In CW from top left: Published beat detections, average bpm on each prediction, running detector node, pedal state( position, direction, beats to play), msg from predictor to drum player.

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 the 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 beat updates our array of observations and

array of predicted beats and other model variables. The main loop acts as a timer to execute beat commands to the player. It reads the current time and compares it to the current predicted beat to be played. If the beat is in the margin of the current time adjusted by the hardware lag, a message is sent to the drum player and we move on the next beat by incrementing a counter. If the predicted beat is late, we discard it and increment the counter. The counter is a member field of the model and it also indicates where the predictions should be inserted. The error margin is solely defined by the frequency of this loop.

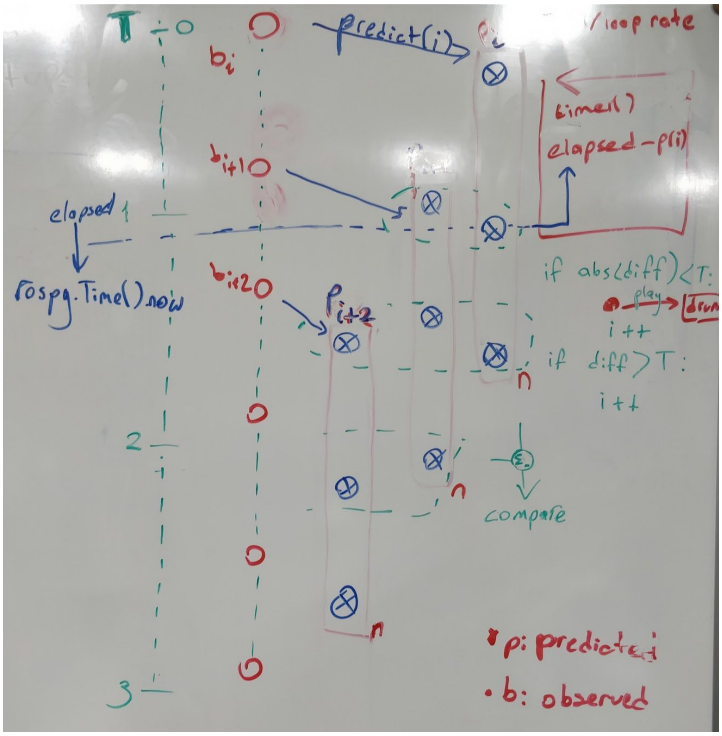



Figure 3: An overlay of a beat_player loop in time domain

The Model class imported from rhythm.py defines the prediction method.
The base class (BaseModel) implements observation updates, prediction and re-prediction whereas the inherited Model class has to implement the fitting function for a given element and update of its model specific variables.

Currently, the running average (*RunningAverageFit*) can handle main beats of bars, whereas *WindowsofN* is able to create patterns of varying intervals by creating a pattern out of recurring intervals. In order to do this, it compares the intervals every "N-plus-i"th element's interval, where N is the length of pattern and i is the index of the subject beat of the said pattern. Then, the model adjusts N according to the RMSE of average intervals versus the actual intervals of the beats. If the intervals actually have a simpler pattern, the algorithm folds them back to a smaller window.

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. All nodes can bypass

the previous steps for easier testing;  this includes but not limited to running the player with only observations instead of predictions, directly playing beats or key presses, or using offline audio or extracted beats.

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 beat issued when the pedal head is too close to the drum, it will only retreat a little only to hit the drum quickly.

Thanks to the processing power of Jetson TX2, we are able to breach 100Hz on our algorithm loops and operate the stepper motor with one complete step a cycle at any given motor speed.

## 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 the 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). We also plan to use a Hidden Markov Model to predict the multivariate pattern and test trained models of RNN on the rhythm prediction step of the software.

Currently, the hardware lag is compensated by a predefined parameter. This can easily be automated 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 playing the assumed model.

There are numerous hardware improvements to be made when selected to attend GTC 2018 in March! Including new gearing with faster action for drum hits and soundproofing the microphone from the drum.