State Estimation of a Quadrupedal Robot using Temporal Convolutional Networks

Ahmed Mohamed

November 3, 2022

University of Duisburg-Essen

Contents i

Introduction

Odometry for wheeled robots

Odometry is essential for autonomous robots, and it is used for the purpose of calculating the robot locomotion parameters such as:

- Speed
- Position
- Orientation

The data from encoders, which track the wheel rotations and steering angles, can be used to compute **wheeled robot** odometry from a specified starting point. The data that the encoders provide is as follows:

- Forward motion rate
- Rotational rate

Odometry for wheeled robots

The robot's state can be described as following:

- (x): Robot's x-axis position
- (y): Robot's y-axis position
- (θ): Rotation angle with respect to the original starting point

Using robot's forward motion rate and rotational rate one can calculate the state of the robot (x, y, θ)

Odometry for wheeled robots

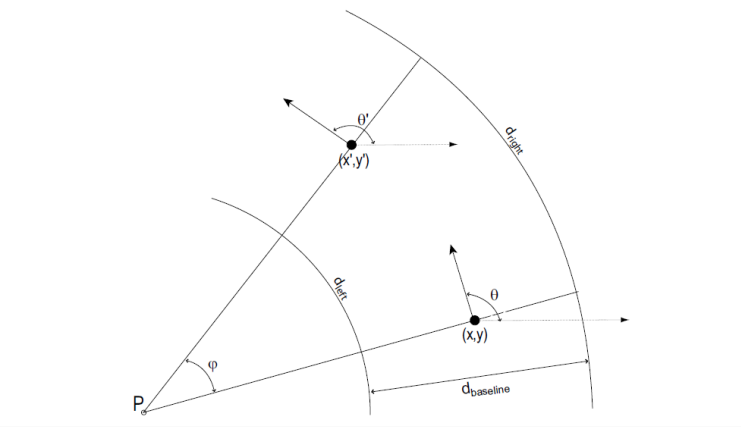


Figure 1: Odometry geometry for wheeled robot

The odometry problem is to solve for $(x^{'}, y^{'}, \theta^{'})$ given (x, y, θ) and $d_{baseline}$. The robot shown in the figure is rotating counterclockwise.

Kinematics analysis of a quadruped robot

Due to the fundamentally different kinematics used for locomotion by legged robots, its odometry cannot be directly determined using the previous method.

The same category of robots share all similar dynamic properties such as: degree of freedoms, number of joints, number of legs, etc, but with differences only in the links length and sensor locations. A1 Unitree robot have the same kinematic tree as HyQ and ANYmal, therefore A1 has the following properties:

- Machine Weight(with battery): 12kg
- Dimensions(Stand): 500×300×400 mmL×W×H
- Dimensions(Folded): 450×300×150 mmL×W×H
- Maximum Walking Speed: 3.3 m/s
- DOFS: 12
- Joint Torque: 33.5 NM
- Joint Maximum Speed: 21 rad/s
- Foot Force Sensors: 4

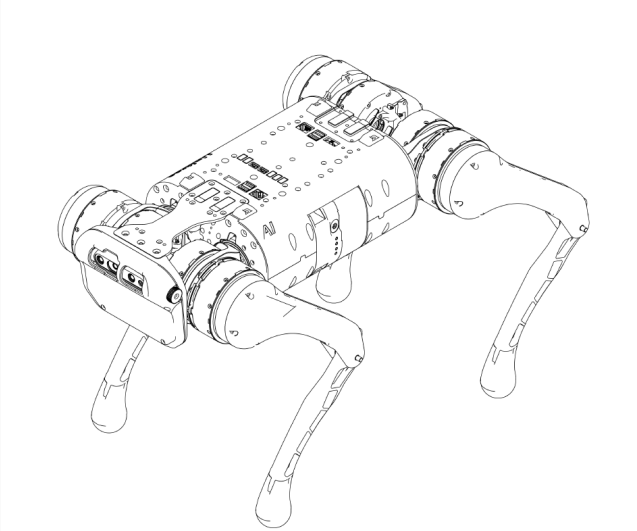


Figure 2: Unitree A1 Quadruped robot

1. Body
2. Front cover
3. Abdominal support pad
4. Foot assembly
5. Hip joint
6. Thigh
7. Knee joint
8. Calf

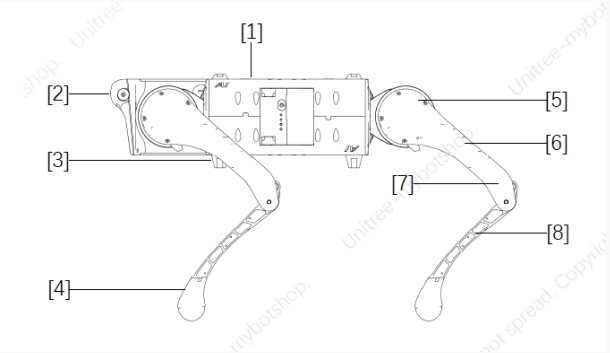


Figure 3: A1 robot part names

Coordinate, joint axis and zero point:
The rotation axis of the hip joint is the x-axis, and the rotation axis of the thigh and calf joints is the y-axis, and the positive rotation direction conforms to the right-hand rule.

Joint limitations

- Hip joint: [-46°, 46°]
- Thigh joint: [-60°, 240°]
- Calf joint: [-154.5°, -52.5°]

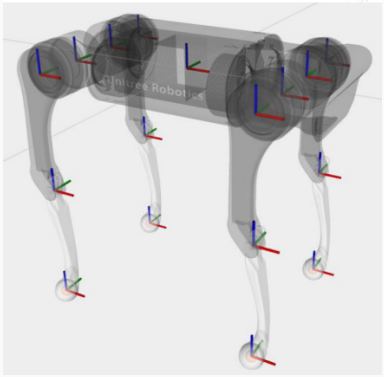


Figure 4: All joint coordinates under ROS

Foot force sensor: There are four force sensors at the same position of each foot. The position is as follows: The blue part in the figure is the sensor, and the red arrow represents the direction of the force sensor. The direction of the force on the sensor is determined by the specific contact form of the foot end and the ground

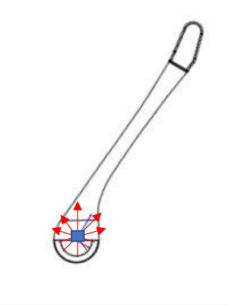


Figure 5: Force directions of the foot sensor

Literature about localization of pedal robots

The following are the two main methods for localizing quadruped robots:

- Localization system based on joint encoders and torque sensors.
- Localization system using visual-inertial odometry (VIO), which combines data from a camera and an inertial measurement unit (IMU).

Problems for odomotery for pedal robots

- Quadruped robots require **fast localization** since control and path planning are more challenging for them than for other mobile robots.
- **Odometry drift** was brought on by slippery or deformable surfaces since they go against the assumption of stable footholds.
- Odometry approaches experience **long-term drifting** in global translation and orientation, regardless of the underlying mathematical formulation on which they are based.
- **The frequencies and delays** in many of the sensors are the fundamental disadvantages of visual odometry methods.
- Localization accuracy decreases because of unsuccessful point feature tracking in **textureless areas** like the ground and sky.
- The robot's movement is causing the camera image to become **blurry**.
- Quadruped robots have **repetitive wobbling** due to the walking process.

1. What is sequence modeling?

Creating a sequence of values out of a set of input values is a machine learning approach known as "sequence modeling." These input values might take the shape of time-series data, which displays how a certain variable changes over time. The output may depend on future data or it may be causal, meaning that there is no leakage from future inputs.

2. What is TCN?

Temporal Convolutional Networks, or TCNs, are a subclass of Convolutional Neural Network (CNNs) that have the following characteristics:

- The output has the same length as the input
- There can be no leakage from the future to the past

The TCN uses a 1D fully convolutional network to accomplish the first point, and causal convolutions to meet the second.

$$TCN = 1D_{FCN} + causalconvolutions$$

Literature about TCN

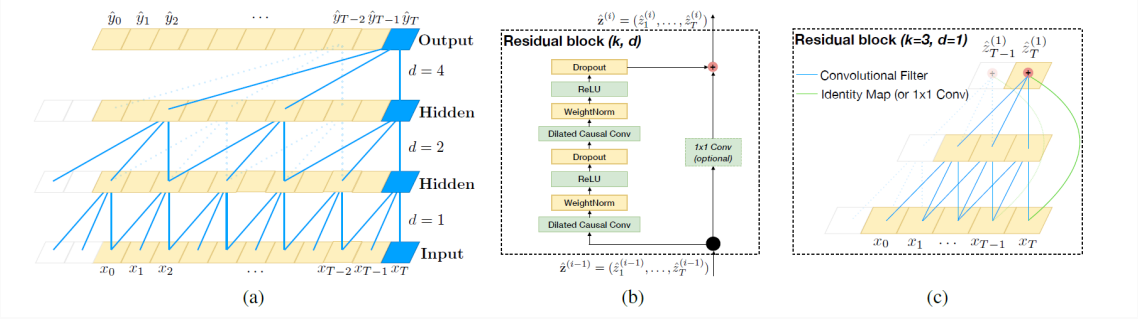


Figure 6: Architectural elements in a TCN

In the previous figure:

- (a): A dilated causal convolution with dilation factors $d = 1; 2; 4$ and filter size $k = 3$. The receptive field is able to cover all values from the input sequence.
- (b): A residual block stacks two dilated causal convolution layers together, and the results from the final convolution are added back to the inputs to obtain the outputs of the block.
- (c): Example of residual connection in a TCN.

Contribution: TCN for Localization of quadrupedal robots

The following is a description of a related paper that utilise Temporal Convolutional Networks to predict the state of the quadruped robot. In [4] Sequence model is used, specifically a temporal convolutional network (TCN), which produces movements based on an extensive history of proprioceptive states, as appeared differently in relation to a multi-layer perceptron (MLP) alone, which operates on a certain snapshot of the robot's current state as was frequently used in related work.

Contribution: TCN for Localization of quadrupedal robots

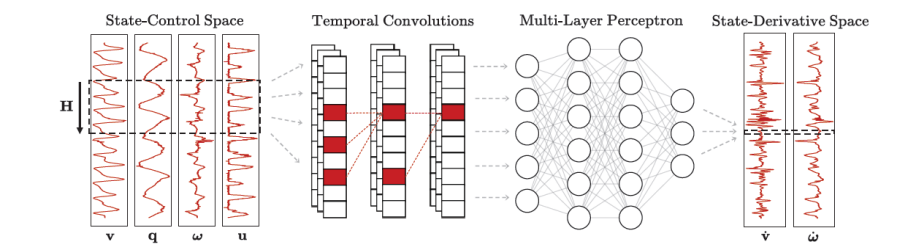


Figure 7: PI-TCN's architecture

A multi-layer perceptron (MLP) and a temporal convolutional network (TCN) are combined in an architecture known as (PI-TCN) which is proposed in this paper as a Physics-Inspired Temporal Convolutional Network.

- The features from the former states' history are encoded by the TCN.
- The TCN representation is decoded by the MLP to produce precise robot state predictions.

Problem representation and suggested approaches

Problem representation

- The following definition of the problem can be made in light of the preceding sections: **How can proprioceptive data of a quadrupedal robot be used to predict its state?**
- This problem falls under the category of a machine learning **regression** problem since the values to be investigated or predicted are continuous, so what is regression?
- **Regression** is a technique or algorithm for figuring out how independent features relate to a dependent result. Once the link between the independent and dependent values has been estimated, outcomes may then be predicted.

Suggested approaches

- Temporal Convolutional Network (TCN) is proposed to calculate the relative position between two proprioceptive sequences. Proprioceptive information is used to produce an odometry result, it includes joint states like velocity, effort, relative position, and foot interactions, which are accumulated along the path traveled.
- A second deep Multilayer Perceptron (MLP) is proposed to decode the TCN proprioceptive representation in order to provide predictions of robot state and reduce sensor inaccuracy, which is described in the preceding sections. The two networks are described in depth in the following sections.

Proposed TCN Architecture

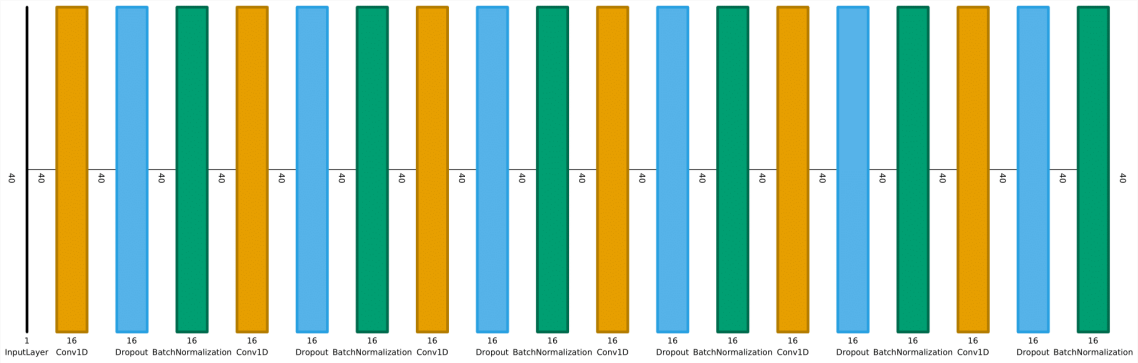


Figure 8: Architecture for Temporal Convolutional Network (TCN)

Proposed TCN Architecture

- The previous figure shows the structure of the TCN network, which has five hidden layers, each of size 16, and a dilation sequence of [1 2 4 8 16 32] to cover as much as possible of the receptive field, which is the history of the proprioceptive data (Input).
- Each layer of the TCN is stacked with a ReLU activation function, batch normalization, and a Dropout regularizer with rate 0.1.

Proposed MLP Architecture

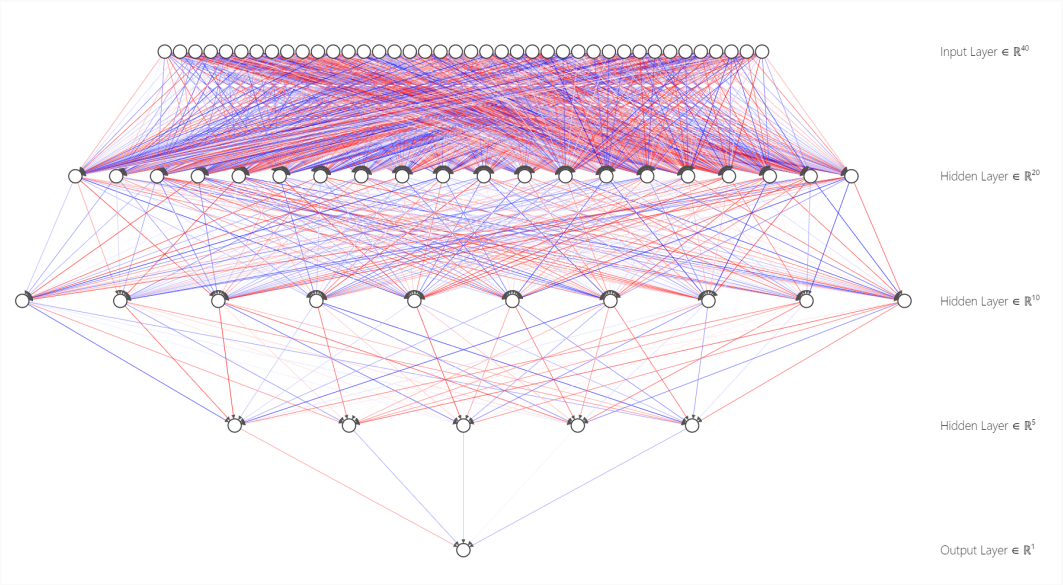


Figure 9: Architecture for Multilayer Multilayer Perceptrons (MLP)

Proposed MLP Architecture

The previous figure shows the structure of the MLP network, which has three hidden layers with sizes: 20, 10 and 5. Each layer of the MLP is stacked with a ReLU activation function.

Dataset and Implementation

Structure and generation of the dataset

The joint states data, which are the data released under the topic /joint_states, are the primary features used for training. Each message is of type sensor_msgs and has the following structure:

- Header header
- string[] name
- float64[] position: the position of the joint (rad or m)
- float64[] velocity: the velocity of the joint (rad/s or m/s)
- float64[] effort: the effort that is applied in the joint (Nm or N)

Each joint is uniquely identified by its name. The header specifies the time at which the joint states were recorded. All the joint states in one message have to be recorded at the same time.

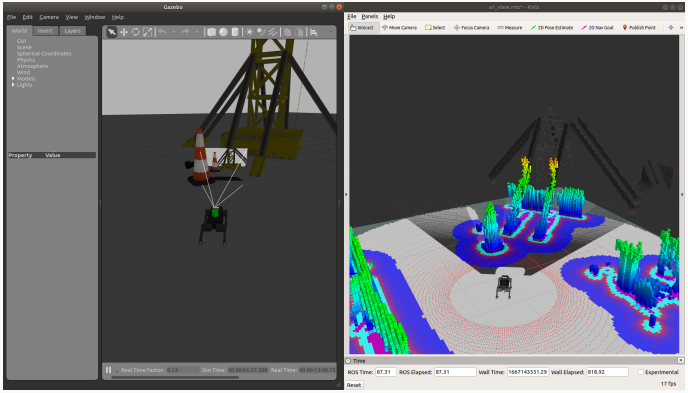


Figure 10: Gazebo and Rviz environment

The data is recorded using a ros package called rosbag and is converted to a csv file using an open source tool created by MIT. It is published as soon as the A1 robot's environment is run in gazebo and rviz.

Data Visualization

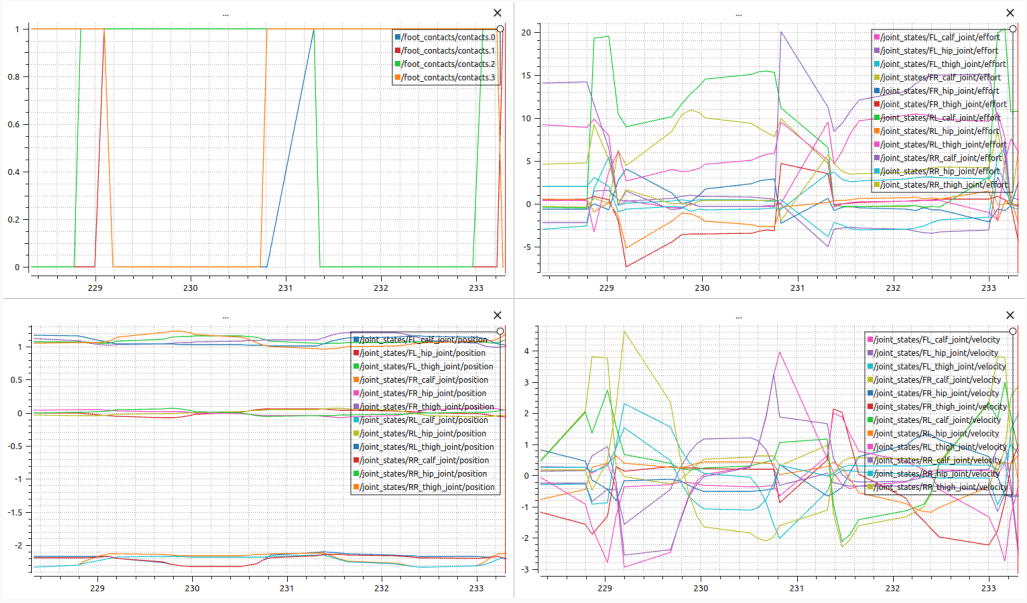


Figure 11: Foot interactions and joint states' changes over time

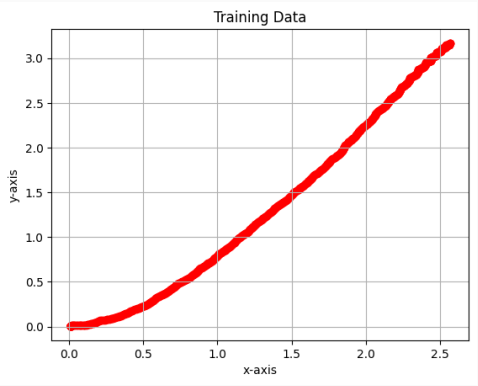
In the previous figure:

- The visualized data is a snapshot taken at a certain time interval that shows how the position of the joints, effort, velocity and foot contacts change over time.
- x-axis is time in seconds
- y-axis is the corresponding values of (position (rad), velocity (rad/s), effort (N)) for each time slot

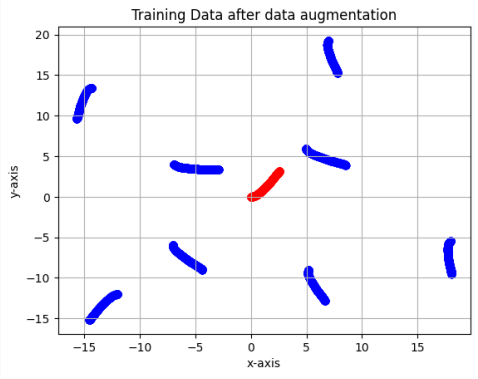
Data augmentation and trajectory generation

In order to enhance the dataset size and prevent overfitting, the following section displays many basic trajectories that were recorded using the ros environment and how these trajectories were rotated around random points with random degrees.

Data augmentation and trajectory generation



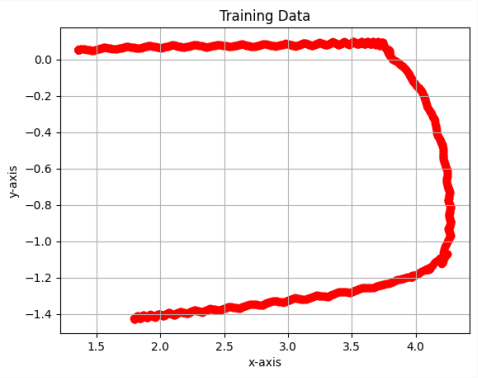
(a) Training dataset



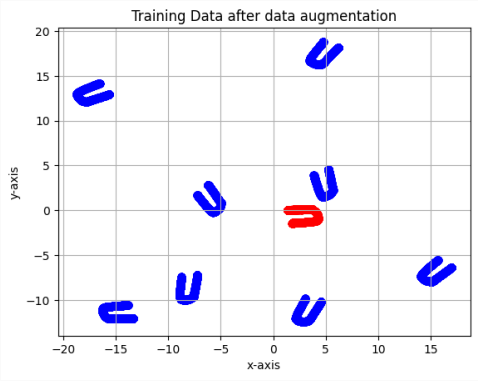
(b) Training dataset after multiple rotations

Figure 12: Augmentation of the training dataset

Data augmentation and trajectory generation



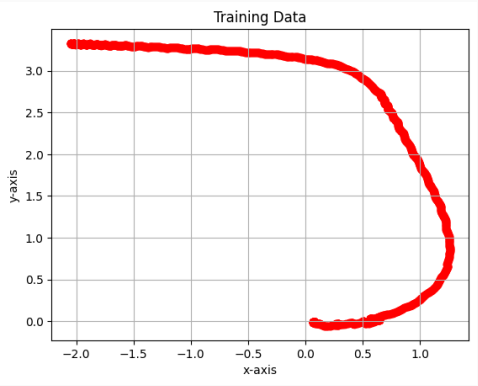
(a) Training dataset



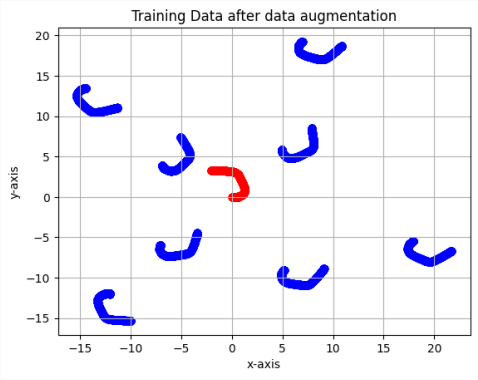
(b) Training dataset after multiple rotations

Figure 13: Augmentation of the training dataset

Data augmentation and trajectory generation



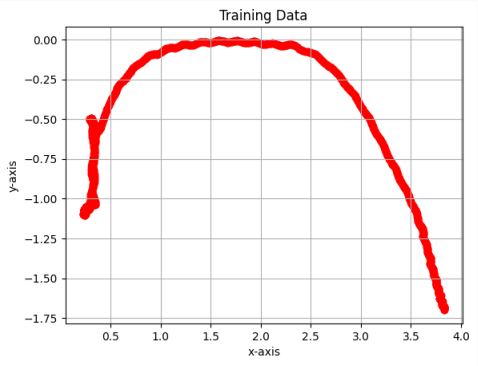
(a) Training dataset



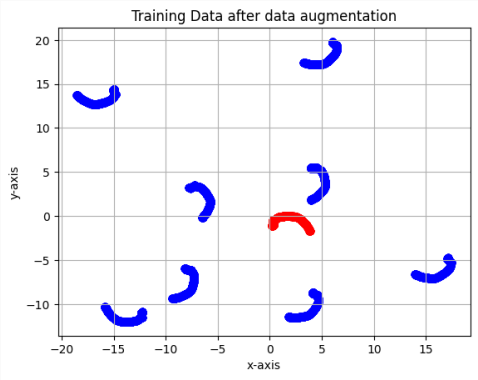
(b) Training dataset after multiple rotations

Figure 14: Augmentation of the training dataset

Data augmentation and trajectory generation



(a) Training dataset



(b) Training dataset after multiple rotations

Figure 15: Augmentation of the training dataset

Already conducted tests

The previously augmented data is provided as input to the network with features F = 40, which represent the history of joint states and foot contacts, and 37200 simulation samples periodically equally spaced with t = 20ms and it has been trained on the simulation dataset for 125 epochs using Adam stochastic gradient descent algorithm, with number of batches $B = 1$ sample, and a constant learning rate of 10^{-4}.

Validation Dataset

- K-fold cross validation is used as a validation dataset given to the network. and as a definition for K-fold cross validation: The dataset is divided into a K number of folds during cross-validation, which assesses the model's performance when faced with new data.
- K is the number of groups into which the data sample is divided.
- In machine learning, cross-validation is frequently used to enhance model prediction
- In the proposed model the number of splits is five ($k = 5$).

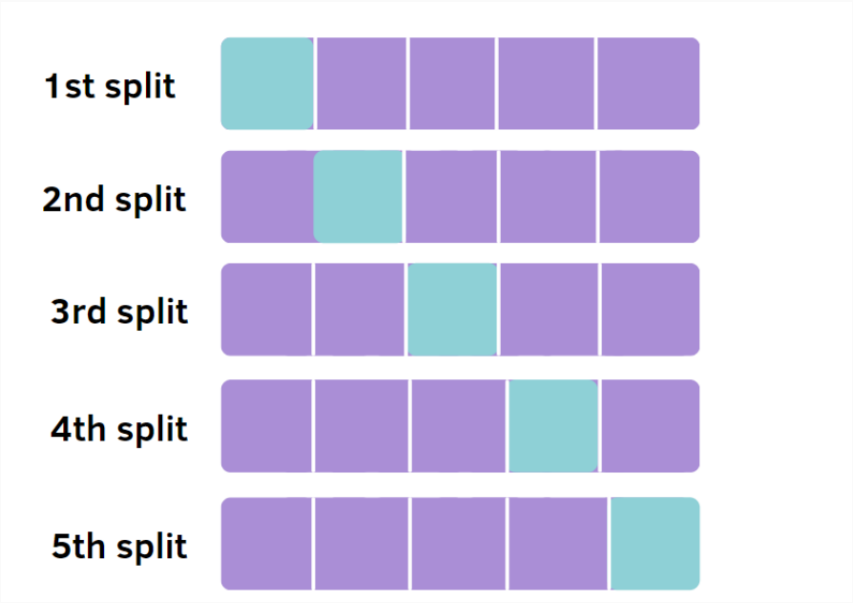
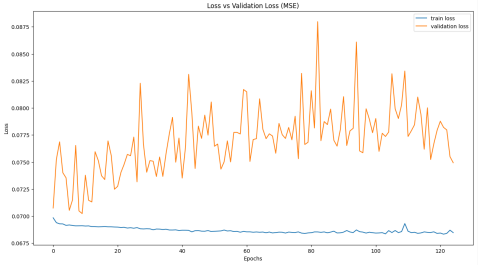
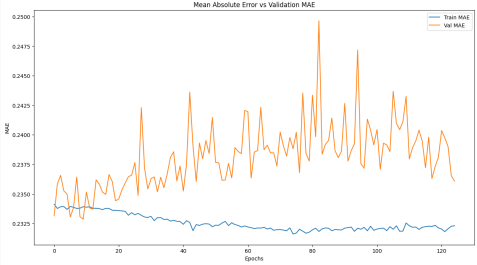


Figure 16: Data's division into training and validation datasets

First results



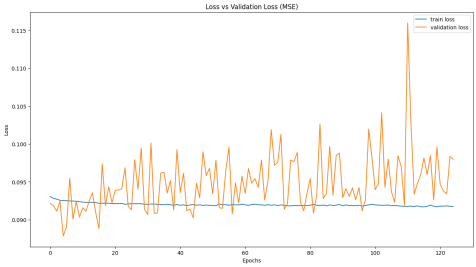
(a) Training vs validation loss (MSE)



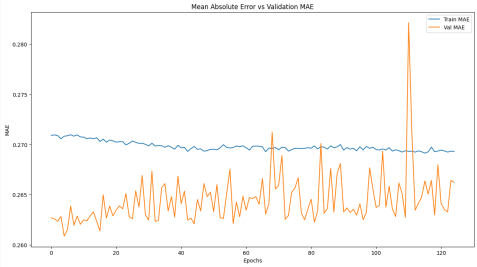
(b) Training vs validation MAE

Figure 17: First results for x positions

First results



(a) Training vs validation loss (MSE)



(b) Training vs validation MAE

Figure 18: First results for y positions

First results

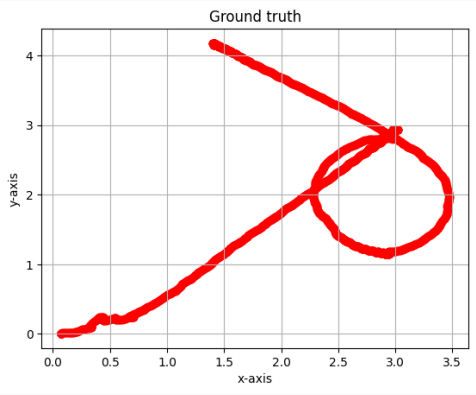
First results in percentage		
Metrics	x-axis	y-axis
MAE	30.4	29.4
Loss(MSE)	12.6	11.8

- MAE: Mean Absolute Error
- MSE: Mean Squared Error

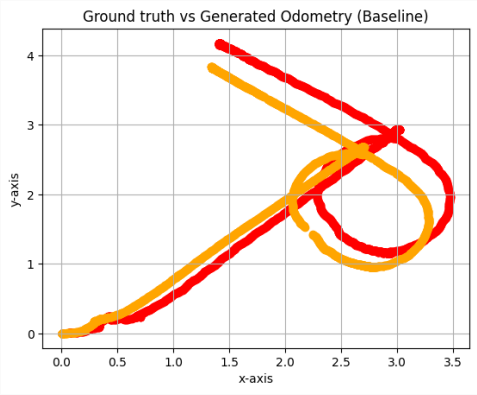
Evaluation

Baseline

- In the previous section, data such as loss and mean absolute error was visualized and presented in percentage, but what do these outcomes mean?
- In order to be able to analyze the results in a good way we need to use a baseline, and simple odometry trajectories produced using the Extended Kalman Filter (EKF) algorithm are the suggested baseline to use because it has already experienced long-term drifting due to sensor delays, inaccuracies, and textureless areas.
- The objective is to surpass the baseline accuracy.



(a) Ground truth of testing dataset



(b) Ground truth vs proposed baseline

Figure 19: Comparison between ground truth and proposed baseline

The baseline has accumulated drift, as seen in the preceding image, which leads to the following accuracy in the x and y axes:

Baseline results in percentage		
Metrics	x-axis	y-axis
MAE	20.5	22.9

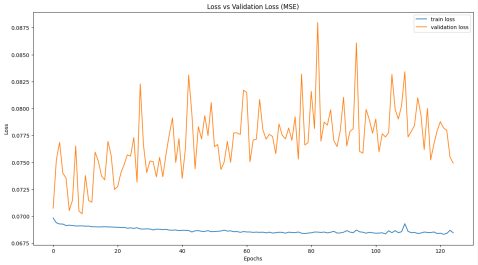
From the above observations, it can be derived that a good model's predictions in both the x and y axes should have a lower mean absolute error than the suggested baseline.

Evaluation results

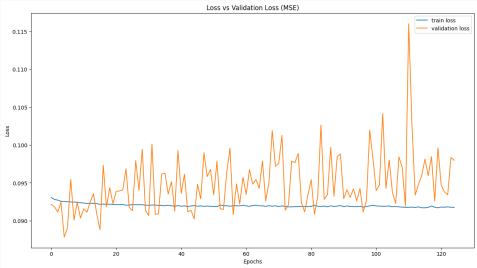
Comparison between baseline and model outcomes				
Metrics	x-axis (Model)	x-axis (Baseline)	y-axis (Model)	y-axis (Baseline)
MAE	30.4	20.5	29.4	22.9

The baseline has superior accuracy in both the x and y axes, according to the observations above, therefore model improvements should be made.

Loss visualization



(a) Loss results for x coordinates



(b) Loss results for y coordinates

Figure 20: Loss results visualization vs Epochs

Loss computation i

- This section will provide an explanation of how the loss in the proposed models is computed.
- Since continuous values must be estimated, as was previously stated in the problem description, the machine learning problem fits under the category of regression. However, in light of the previous concept, another one called autoregression could be presented.
- A regression of the variable against itself is what is meant by the phrase "autoregression" To put it another way, the estimated value for a variable depends on its preceding values.
- To implement the idea of autoregression in the proposed model, the loss visualization in the preceding graph was produced using the following approach.

Loss computation ii

- Since each sample represents an x or y position, the model must calculate the loss and update its weights after each sample because the next position depends on the previous one.
- The proposed model uses a mean squared error function as the loss function, and its formula is as follows:

$$\sum_{i=1}^{D}(x_i - y_i)^2$$

where (D) is number of samples (x) is the predicted value, (y) is the ground truth value.

- The number of samples that are propagated in the network is represented by the batch size in the proposed model, and the loss is calculated after the propagation of each batch.

Loss computation iii

- The batch number of the network is adjusted to 1 in order to perform the autoregression in the proposed model, which requires the model to compute the loss after each sample.

References

[1] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[2] Geoff Fink and Claudio Semini. Proprioceptive sensor fusion for quadruped robot state estimation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10914–10920. IEEE, 2020.

[3] Haitao He and Xiaoping Chen. A model-based approach to calculating and calibrating the odometry for quadruped robots. In *Robot Soccer World Cup*, pages 337–344. Springer, 2007.

[4] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

[5] Hyunjun Lim, Byeongho Yu, Yeeun Kim, Joowoong Byun, Soonpyo Kwon, Haewon Park, and Hyun Myung. Walk-vio: Walking-motion-adaptive leg kinematic constraint visual-inertial odometry for quadruped robots. *arXiv preprint arXiv:2111.15164*, 2021.

[6] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022.

[7] Edwin Olson. A primer on odometry and motor control. *Electronic Group Discuss*, 12, 2004.

[8] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[9] Alessandro Saviolo, Guanrui Li, and Giuseppe Loianno. Physics-inspired temporal learning of quadrotor dynamics for accurate model predictive trajectory tracking. *arXiv preprint arXiv:2206.03305*, 2022.