Setting up a flexible framework to add MPPI support for different dynamics models

Undergraduate Summer Student Final Presentation

Rodrigue de Schaetzen

August 17th, 2020

Outline

- ➢Problem statement
- ➢Proposed Approach
- ➢Related Work
- ➢ML Pipeline
- ➢Vision System
- ➢What still needs to be done
- ➢Conclusion

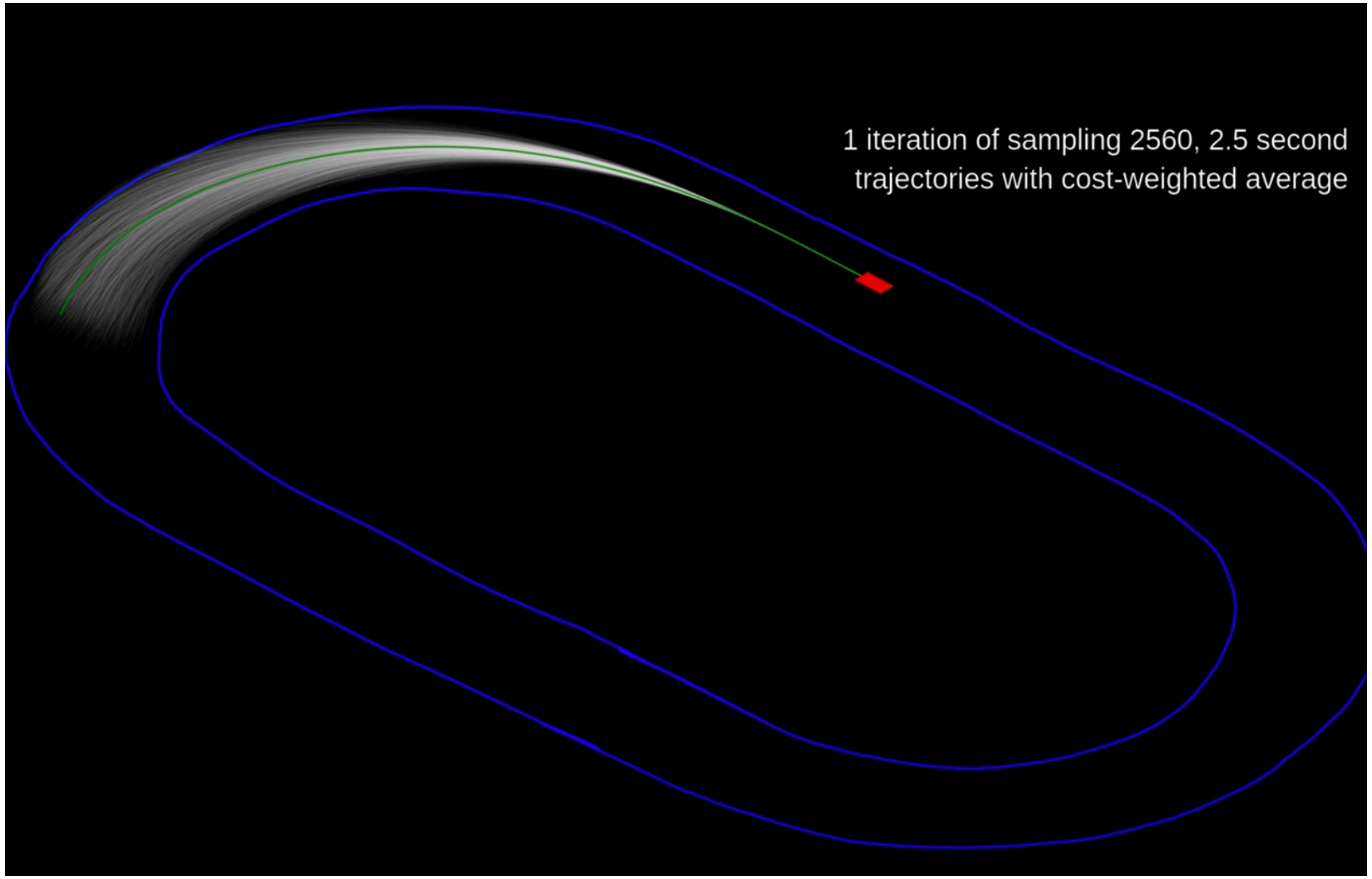
Problem Statement

- The AutoRally platform, an autonomous vehicle testbed for high-speed aggressive driving, was designed with a single vehicle model in mind
- No standard approach available to make AutoRally platform compatible with other robots (e.g. wheelchair, jetracer)
- Focus is on the AutoRally control algorithm: MPPI

Picture taken from https://autorally.github.io/

Model Predictive Path Integral (MPPI)

- Novel approach for autonomous vehicle control
- Based on stochastic sampling of trajectories
- Can handle complex non-linear dynamics and cost functions
- Requires a model of the system dynamics (i.e. what is the vehicle's future state if I apply 50% throttle and 20% steering)



MPPI description and screenshot taken from https://autorally.github.io/

Proposed Approach

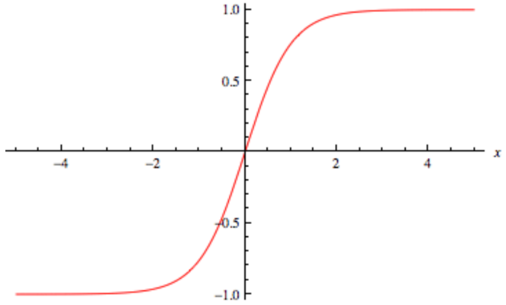
- Build a robust and scalable framework for generating models of different vehicle dynamics in order to add MPPI support for additional robots
    1. ML pipeline for training a neural network dynamics model
    2. Overhead vision system to collect ground truth data
- Already provided is the neural network dynamics model of the AutoRally vehicle – this will serve as a benchmark

Related Work

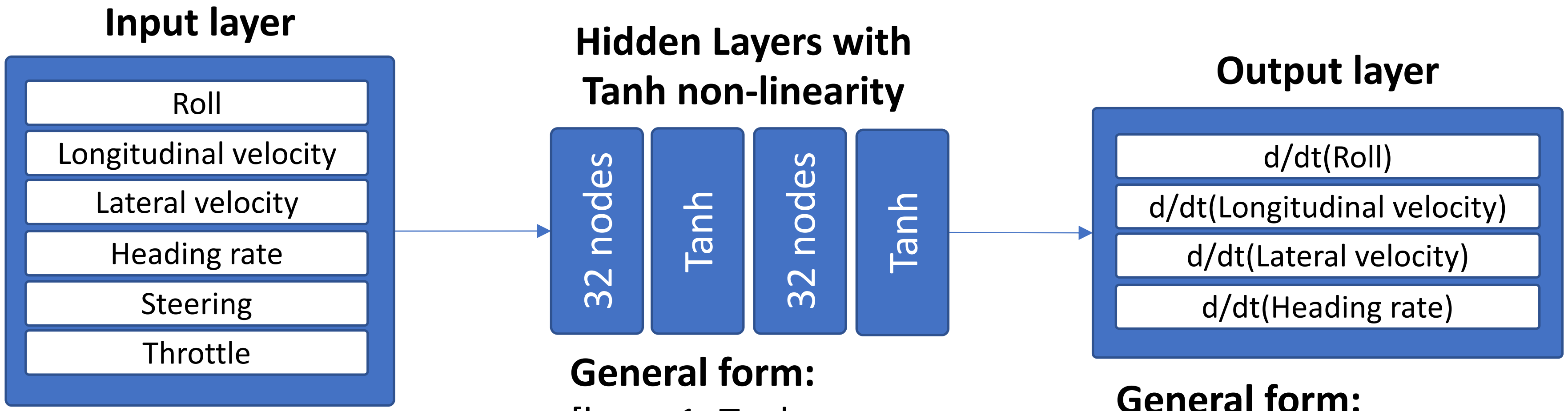
- Approach is built off the ICRA 2017 paper "Information Theoretic MPC for Model-Based Reinforcement Learning"
- Experiment used the AutoRally vehicle
  - Used neural network with only 2 hidden layers each with 32 neurons and Tanh non-linearities as the activation functions
  - Trained on 30 minutes of human-controlled driving around an elliptical track
  - Fused sensor data (GPS, IMU) to get truth pose estimates
- No publicly available code related to model training or testing

Neural Network Dynamics Model

Tanh(x)



- Neural network architecture [6, 32, 32, 4]



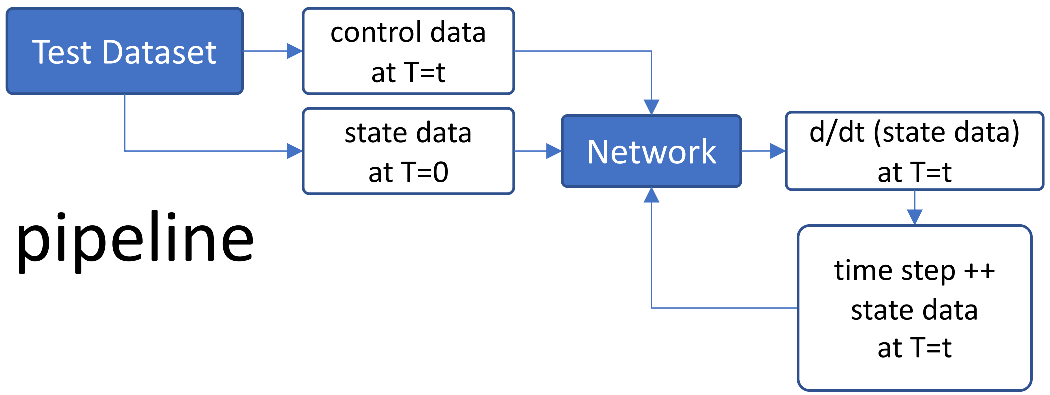
General form:
[state variables,
control variables]

General form:
[layer 1, Tanh, …,
layer n, Tanh]

General form:
d/dt (state variables)

ML pipeline

Generating future states



- Built a highly configurable end-to-end ML pipeline

Data Preprocessing
- Resample state and control data
- Compute state derivative data
- Convert quaternions to Euler angles

Tools: scipy, numpy, pandas

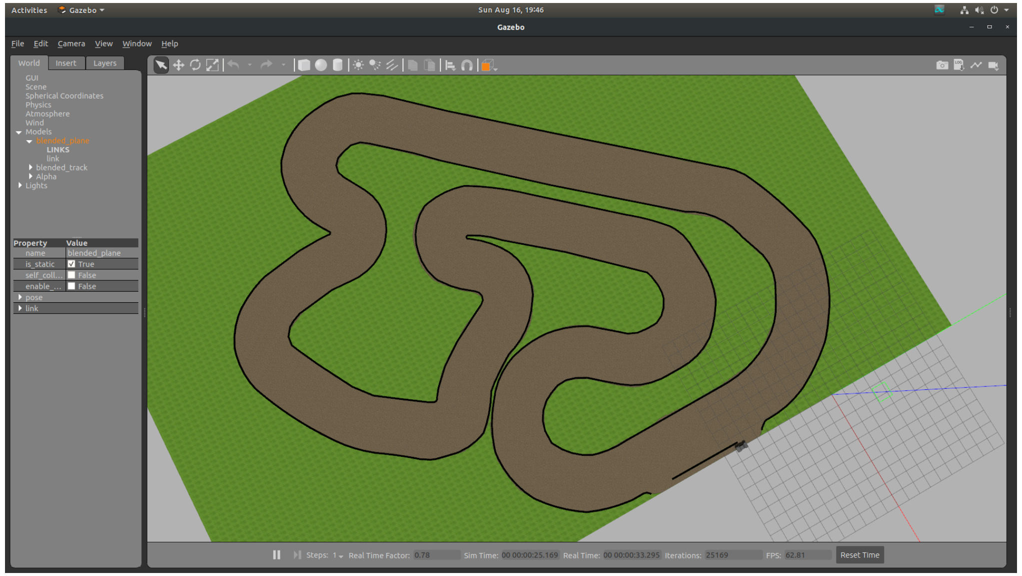
Train Model
- Train feedforward neural network
- Save to disk model with lowest validation loss

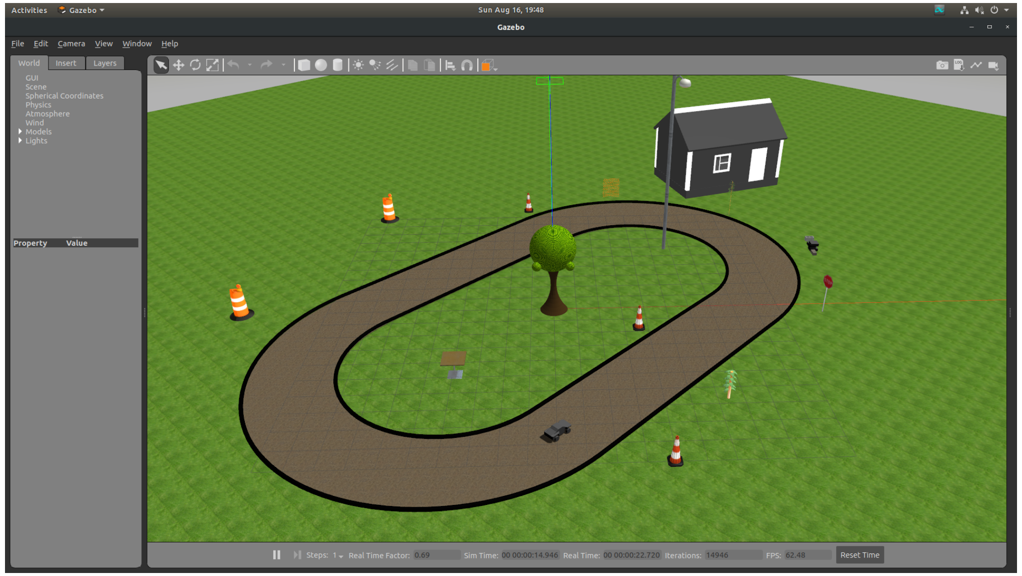
Tools: scikit-learn, pyTorch

Test/Evaluate Model
- Generate future states by feeding current state + controls into the model and using model output to update state
- Compare predicted states to truth states (multi-step error)
- Compute instantaneous errors

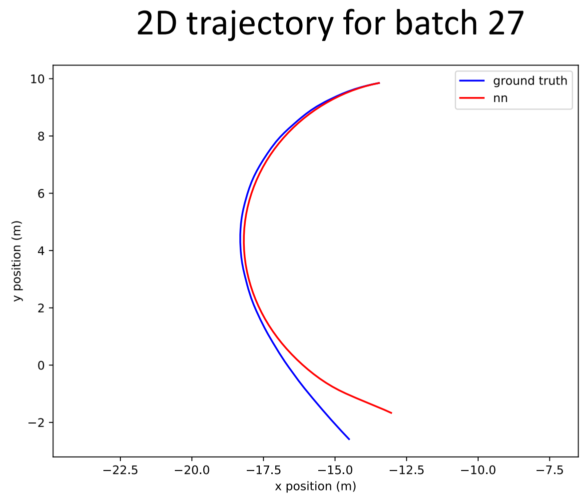
Preliminary Results

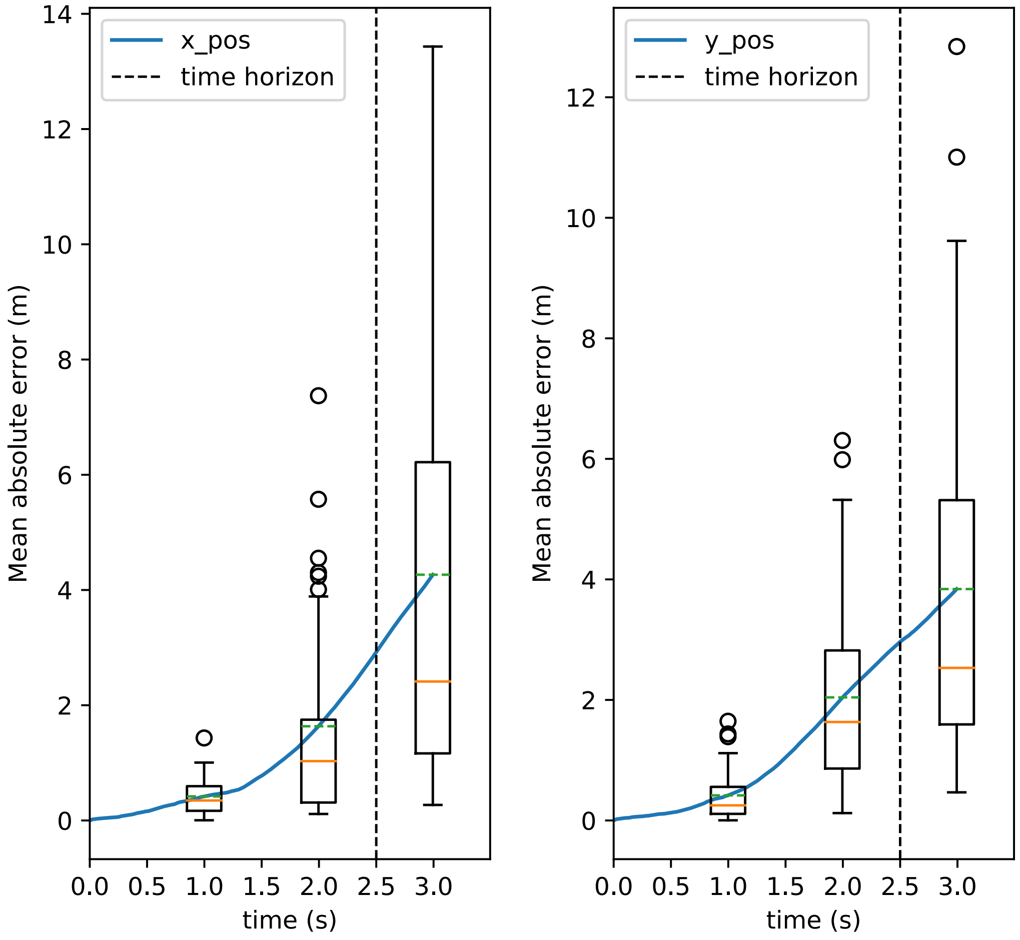
- Tested pipeline with gazebo simulation data
- Ran MPPI and recorded the following data:
  - Train dataset:
    - 15 minutes
    - Elliptical and CCRF tracks
    - Clockwise and counterclockwise
    - 5 m/s and 7 m/s target speed
  - Test dataset:
    - 2 minutes
    - Elliptical track
    - Clockwise and counterclockwise
    - 6 m/s target speed
- Tested original AutoRally neural network as well as some deeper and wider networks
- Ran MPPI with trained model





Preliminary Results





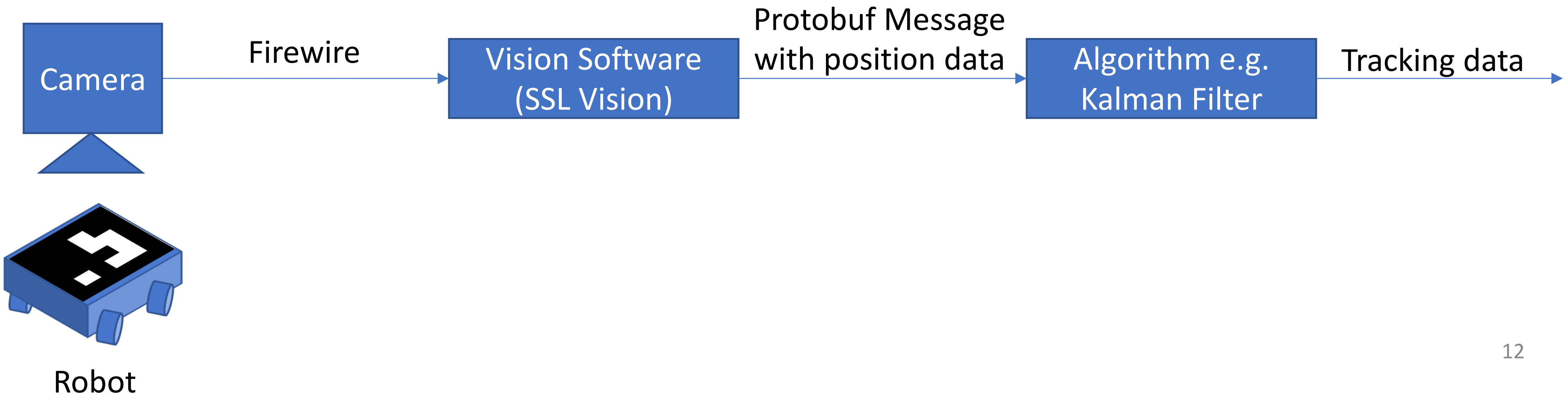
- Test results of a trained model with layers [6, 32, 32, 4]
- Plot shows multi-step prediction error
- Errors from propagating dynamics for 38 different batches

Preliminary Results

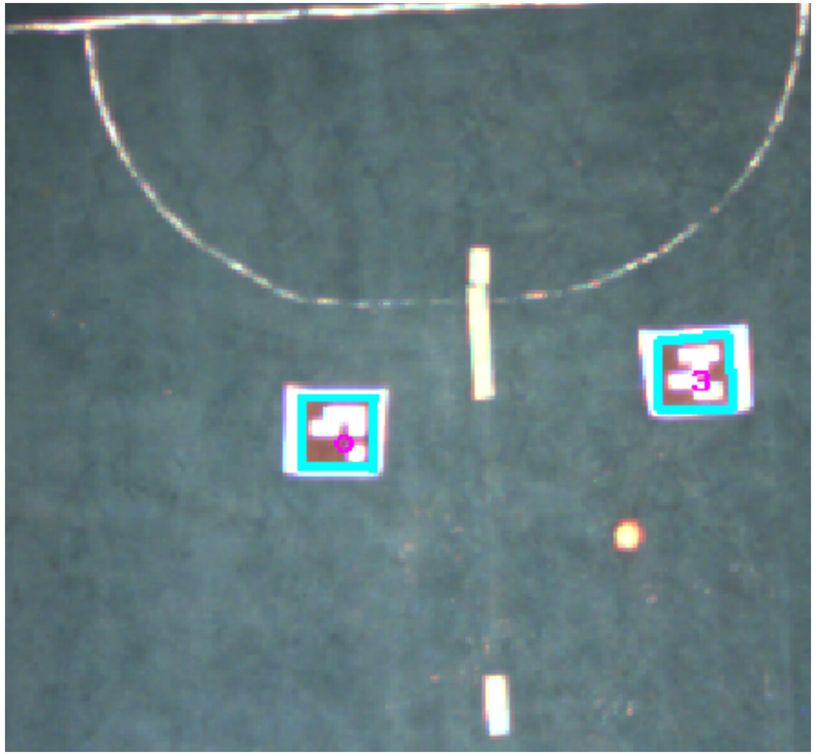
Model name	Network Layers	Mean absolute multi-step errors		
		X position (m)	Y position (m)	Yaw (rad)
AutoRally_nnet (benchmark)	6, 32, 32, 4	2.55 m (σ=2.15)	2.33 m (σ=1.66)	0.37 rad (σ=0.38)
Same-layers (previous slide)	6, 32, 32, 4	2.94 m (σ=3.16)	2.98 m (σ=2.34)	0.93 rad (σ=0.69)
Wider-deeper	6, 64, 64, 64, 64, 4	1.99 m (σ=2.13)	1.62 m (σ=1.24)	0.57 rad (σ=0.55)

Vision system for truth state data

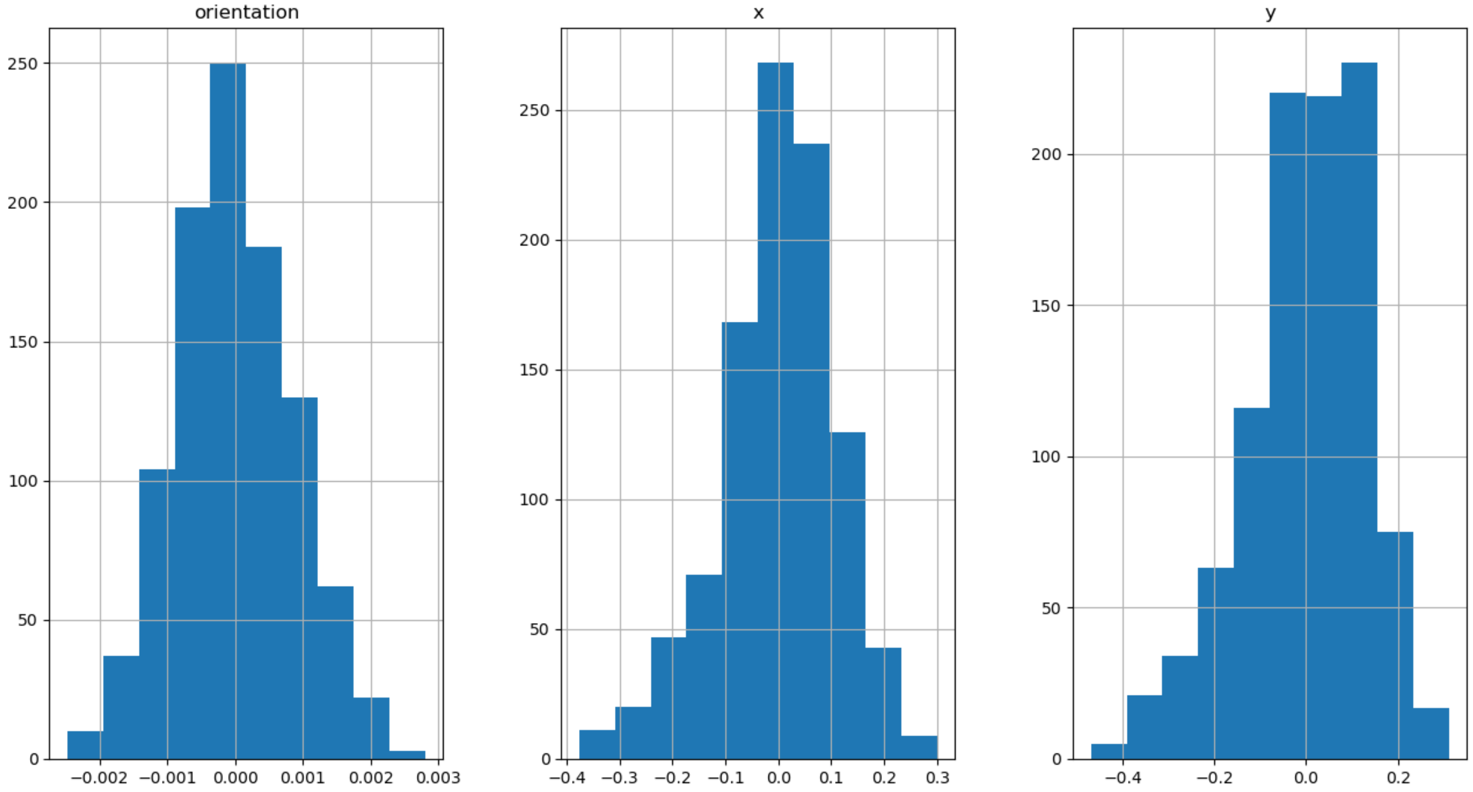
- Need truth data for robots running in the real world
- SSL (Small Size League) Vision can detect robots using overhead cameras
  - Robots are detected via fiducial markers
  - Global x and y coordinates, and robot orientation are computed
- SSL Vision state data can feed to an algorithm such as a Kalman filter to output tracking information (i.e. linear and angular velocity)



Preliminary Results



Histogram of robot position measurements with mean centered at 0
Measurement count=1000



Difference from mean (mm for x,y, and rad for orientation)

Still need to do

- Further quantify sensor noise
- Test vision system on small robot or find larger space
- Use ML pipeline to train model with real world vehicle data
- Modify AutoRally MPPI to be more flexible and configurable to handle a wider range of robots (e.g. omni-drive robots)

Conclusion

- **Problem statement:**
    - AutoRally platform and MPPI officially only support AutoRally vehicle
- **Accomplished so far:**
    - Built a scalable ML pipeline to generate a neural network dynamics model
    - Trained on AutoRally gazebo simulation world to validate pipeline
    - Configured an overhead vision system to collect real vehicle truth state data
- **What still needs to be done:**
    - Test pipeline with real robot data!
    - Remove coupling to AutoRally vehicle in MPPI and AutoRally platform

Links to project files

- Forked AutoRally GitHub repository
- ML pipeline directory
- SSL Vision directory