Improving Mobile Robot Perception Using a Kalman Filter and Machine Learning

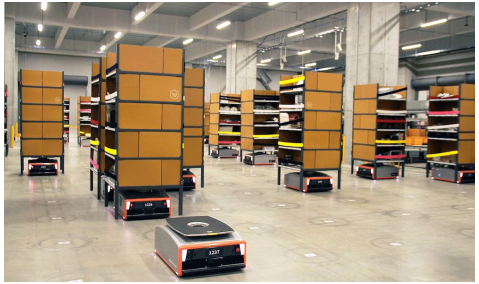
Kevin Robb with Dr. Dean Hougen

Intro to Autonomous Mobile Robots

A “Mobile Robot” is a piece of technology that is able to move around and traverse the environment on its own. The robot will need sensors to perceive and interpret the world.

Examples:

- Self-driving cars
- Warehouse stocking robots
- Floor cleaners at grocery stores or your own home!
- Drones that explore and map sewer systems, forests, etc
- Ground-based search and rescue



GreyOrange warehouse robots.

Minimum requirements for all of these:

- Safe (doesn’t harm itself or others)
- Functional (able to achieve goal)
- Most importantly, able to understand the environment

Importance of Perception

A robot needs to know things about the world to be able to move around and interact with it:

- Where is it?
- Where are environmental features (e.g., obstacles)?
- How can it reach its goal?

These each represent one or more of the following major tasks:

- Deliberative:
  - Mapping
  - Localization
  - Path Planning
  - Navigation
- Reactive:
  - Collision Avoidance
  - Collision Recovery

“Where am I? Where are we going?”

Simple Agent

We need some platform on which to test our improvements to mobile robot perception.

- We use a simulated robot moving in a deliberative way through a random environment.
- Start/End points are always the same, but intermediate waypoints and obstacles are randomized each run.

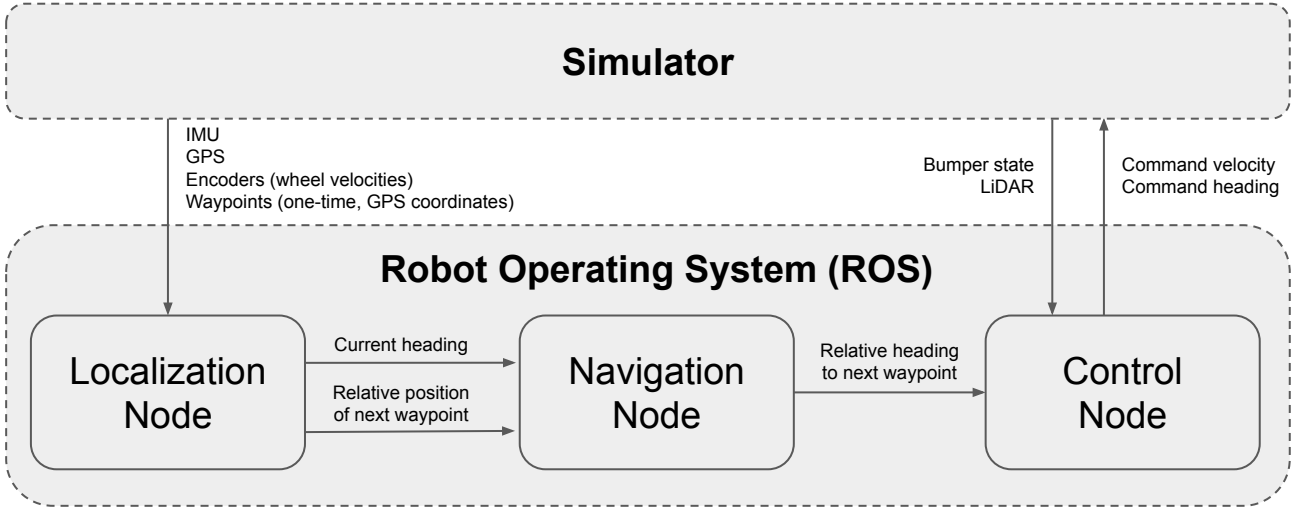
Functions of Agent:

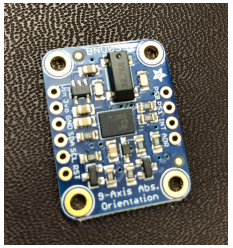
1. Generate data for analysis.
2. Send data to the KF, and receive data back to affect its behavior.

We are not concerned with the actual success of the agent as long as it satisfies these goals.

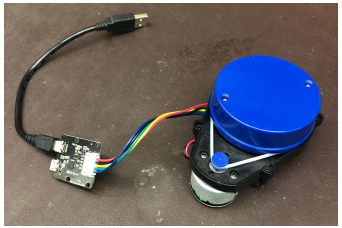
Random run in the simulator. Red = obstacles. Yellow = waypoints. Simulator created by Noah Zemlin for Sooner Competitive Robotics.

Architecture of the Simple Agent





We get the robot's yaw (heading) from the Inertial Measurement Unit (IMU)

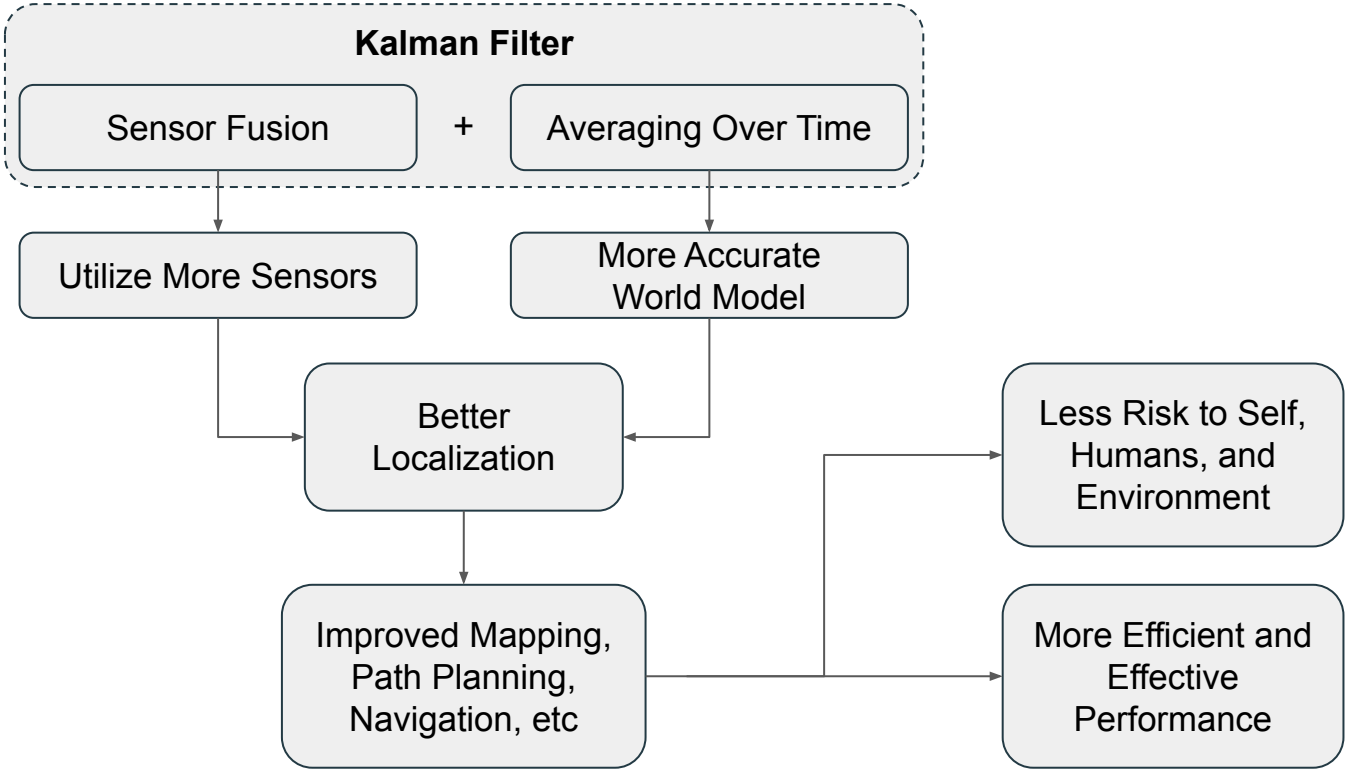


LiDAR is a light-based scanner that gives distances to obstacles in a full circle around the source.

Behavior:

- Directly pursues heading to next waypoint.
- Proportionally increases speed as it gets more aligned with target heading.
- Reactively avoids obstacles with the bumper and LiDAR.

First Method to Improve Perception

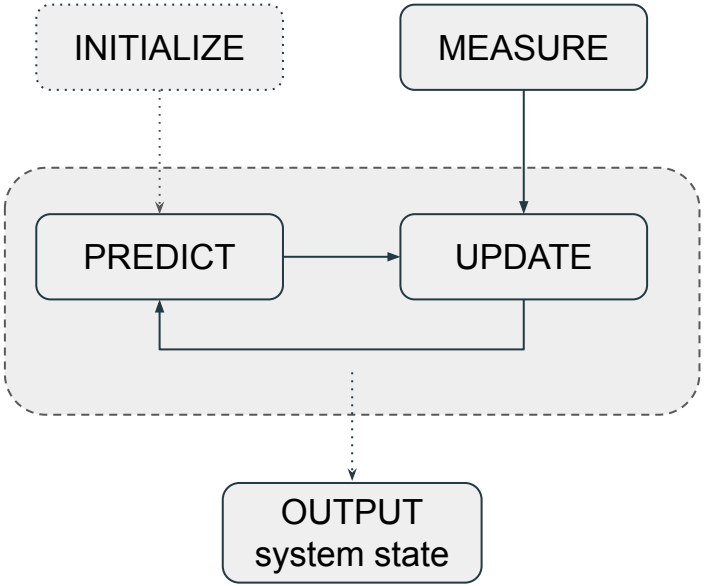


Intro to the Kalman Filter (KF)

Iterative sensor-fusion algorithm that keeps track of an internal model of the robot's state, and updates it with new measurements at each timestep.

- Like a weighted average-over-time
- Tracks uncertainties & covariance as well
- Measurements, ***Z***, include GPS, IMU, encoders
- Uses a “dynamic model” to form predictions

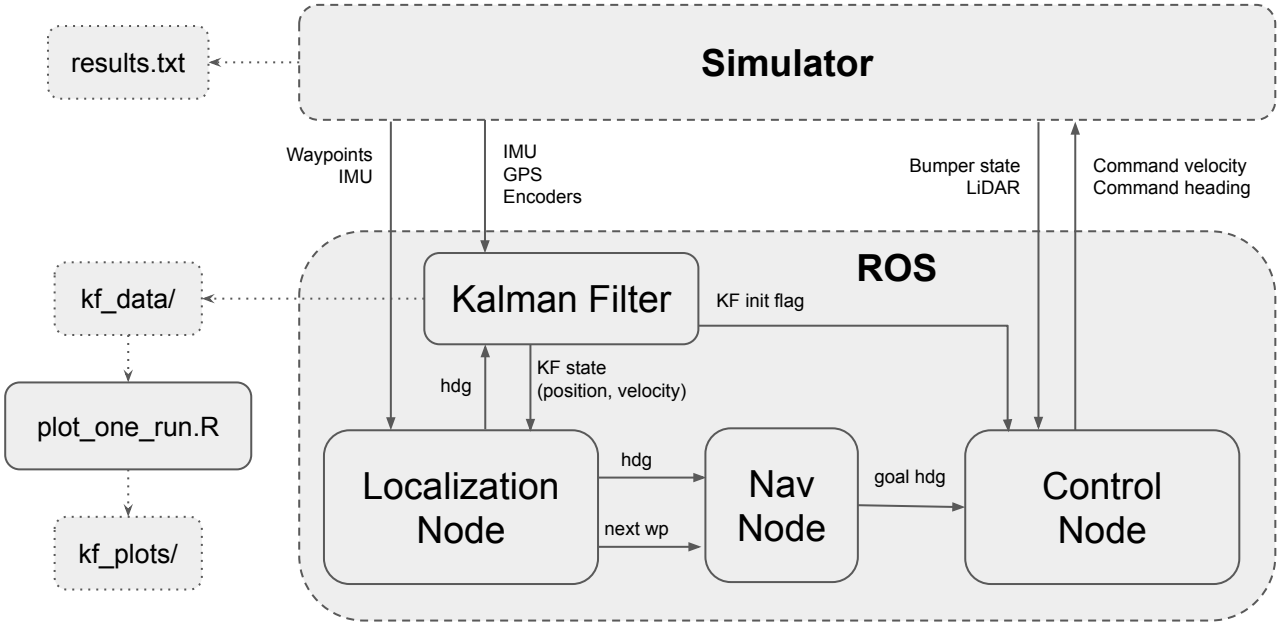
We want to track position (*x* and *y*), and need the velocities to do that; our state is represented by ***X***.



$$Z = \begin{bmatrix} \lambda \\ \Lambda \\ v_x \\ v_y \end{bmatrix} = \begin{bmatrix} (\lambda - \lambda_{start}) \cdot \text{lat\_to\_m} \\ (\Lambda - \Lambda_{start}) \cdot \text{lon\_to\_m} \\ v \cdot \cos\phi \\ -v \cdot \sin\phi \end{bmatrix}$$

$$X = \begin{pmatrix} x & y & \dot{x} & \dot{y} \end{pmatrix}^T$$

Architecture with the KF Integrated

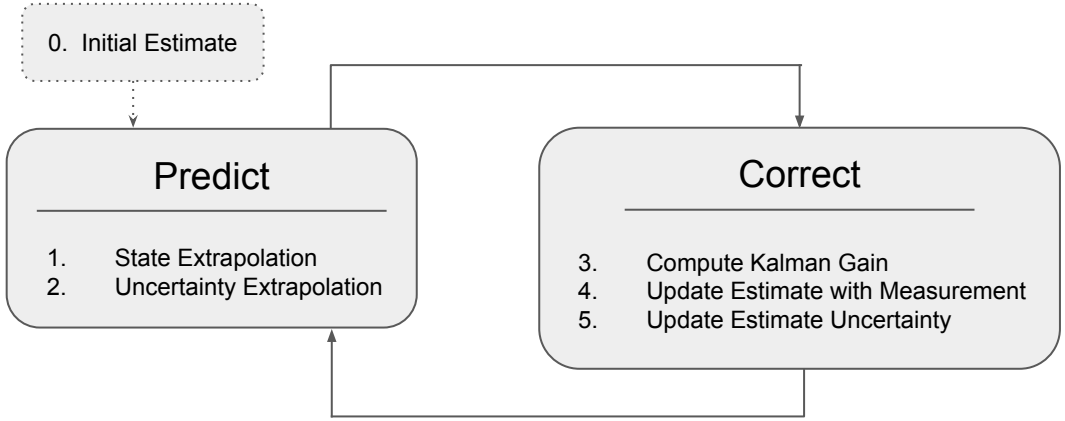


Kalman Filter Process

We separate the KF cycle into two parts: Time Update (Predict) and Measurement Update (Correct).

- Make a prediction.
- Compare this prediction to new measurements.
- Update the state.

Each step in the figure represents one of the five KF equations.



Multidimensional Kalman Filter equation cycle.

$$\begin{cases} \hat{x}_{n+1,1} = \hat{x}_{n,n} + \hat{\dot{x}}_{n,n} \cdot \Delta t \\ \hat{y}_{n+1,1} = \hat{y}_{n,n} + \hat{\dot{y}}_{n,n} \cdot \Delta t \\ \hat{\dot{x}}_{n+1,1} = \hat{\dot{x}}_{n,n} \\ \hat{\dot{y}}_{n+1,1} = \hat{\dot{y}}_{n,n} \end{cases}$$

Dynamic model used for state extrapolation (prediction).

$$\boldsymbol{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

State transition matrix that encodes this model.

Hyperparameters of the Kalman Filter

Some parts of the KF are tuned beforehand by the user for a certain environment. We do this manually for now, but later improve them with machine learning.

$$\boldsymbol{P}_{initial} = \begin{pmatrix} 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0 & 0 & 0.25 & 0 \\ 0 & 0 & 0 & 0.5 \end{pmatrix}$$

*Covariance matrix **P** tracks confidence in our own estimates.*

$$\boldsymbol{Q} = \boldsymbol{R} = \begin{pmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0.01 \end{pmatrix} = 0.01 \cdot \boldsymbol{I}$$

*Process noise **Q** and measurement uncertainty **R**.*

P increases during Time Update and decreases in Measurement Update, so there's a net increase if we don't get confirmation that the estimates are good.

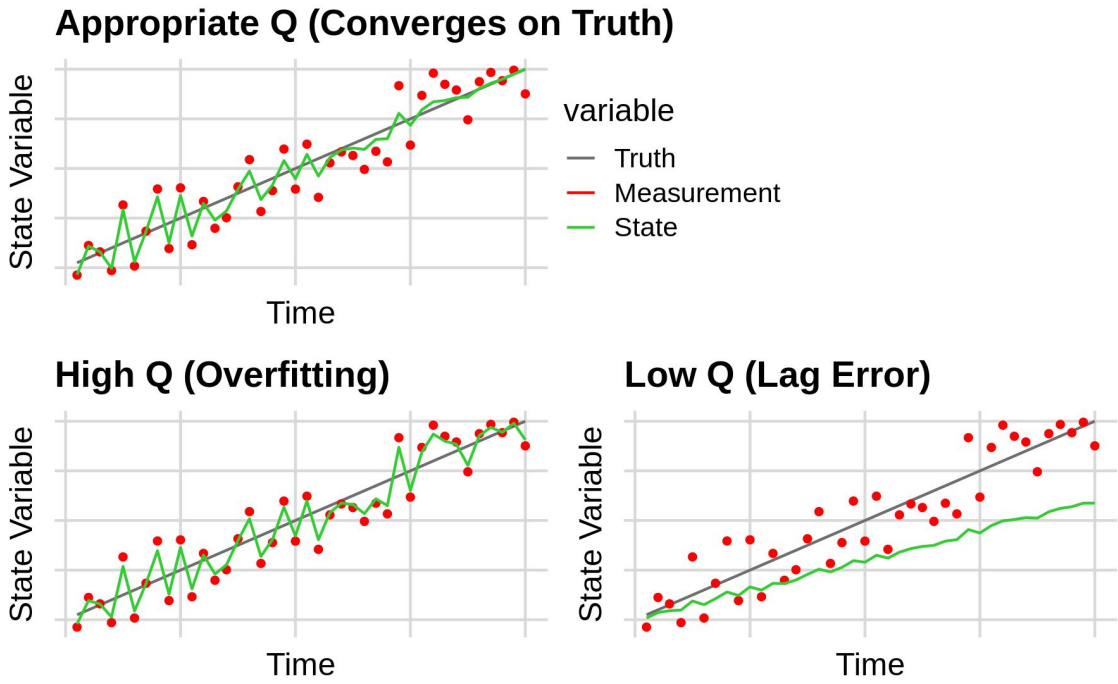
Q and ***R*** do not change.

Importance of the Process Noise, *Q*

Represents the uncertainty in the equations and estimations themselves (the “process”).

Mimics unpredictable deviations from the Dynamic Model.

- e.g., we assume constant velocity, but it’s obviously changing.



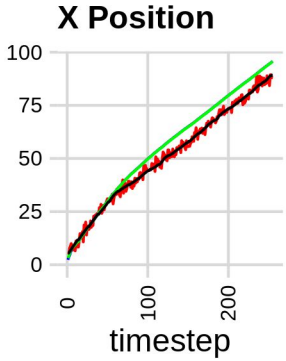
Example behavior of a 1D KF with a constant dynamic model for different magnitudes of Q.

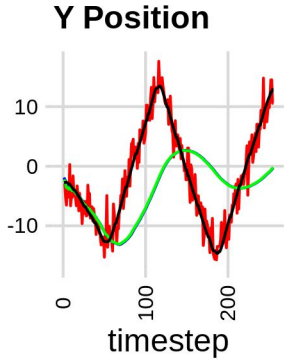
KF Performance (Poor Tuning)

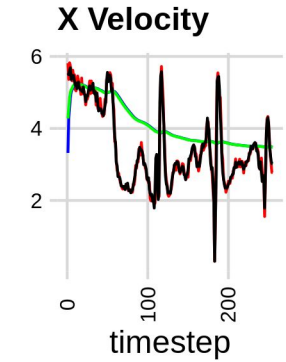
KF follows the trend but lags behind.

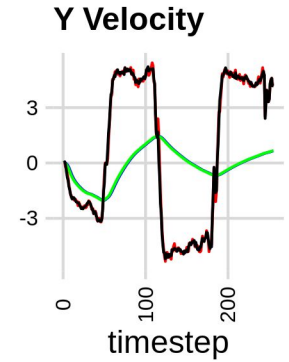
Not aggressive enough to stay on the ground truth, and isn't weighing measurements as much as it should.

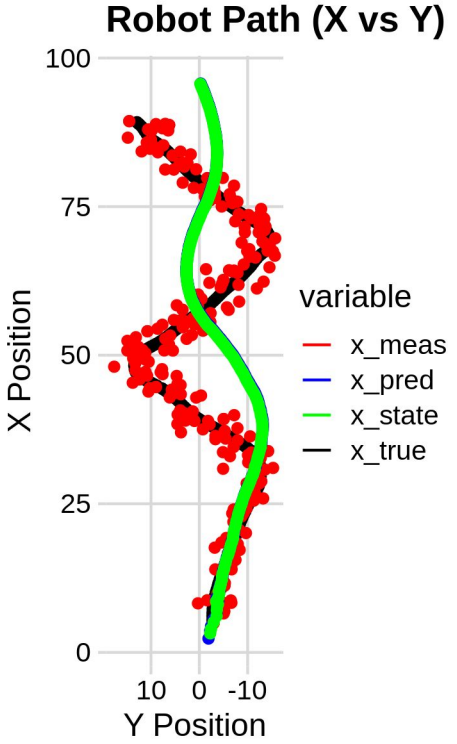
Caused by low **Q** magnitude, leading to more confidence in predictions than is warranted.









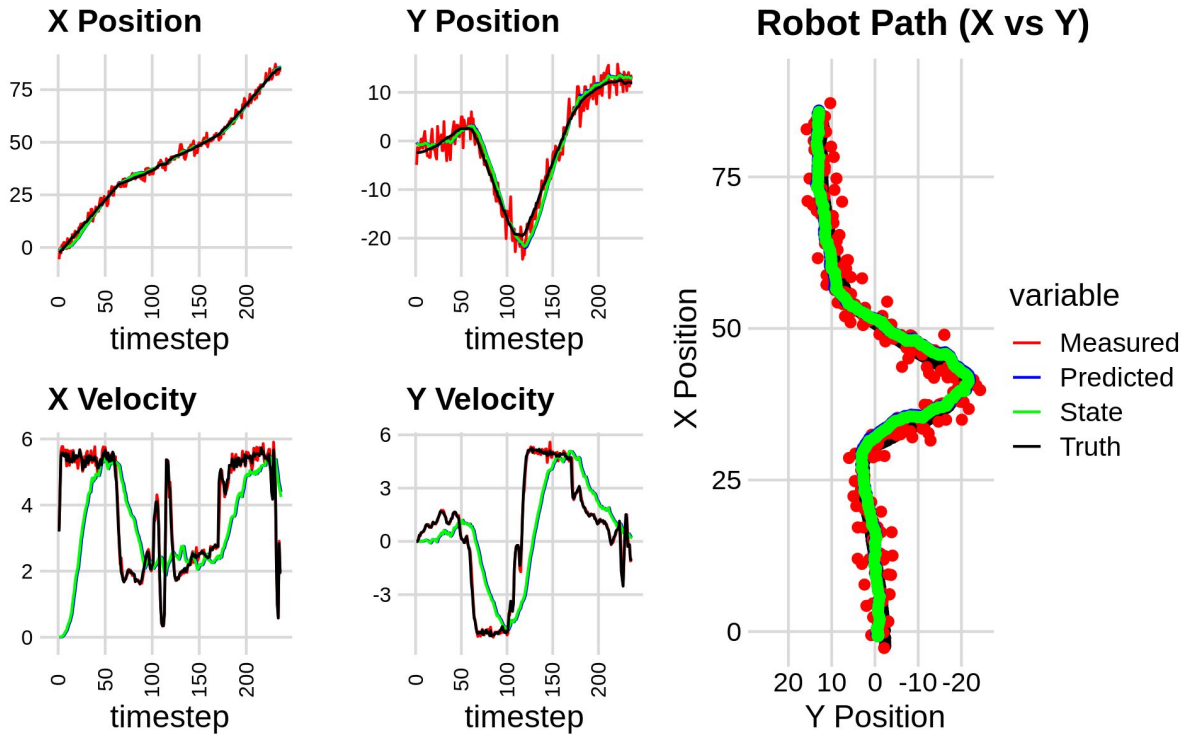


KF Performance (Good Tuning)

KF tracks the position much better.

Still unable to track velocities very well, but this is expected.

- Constant velocity model: KF assumes all changes to velocity are unintentional noise.
- The highest order of the state will be imprecise with this style of model.



We see the position converges on straight segments, and shoots ahead on sharp turns. This represents the steady tradeoff between reliance on measurements vs predictions.

Intro to Evolutionary Computation

- Genetics-based form of machine learning.
- “Agents” each defined by a genome (containing entries of ***P***, ***Q***, and ***R***).
- Evolution of agents over many generations.
- “Survival of the Fittest” determines reproduction.

We can use EC to tune the KF hyperparameters and optimize performance.

1. Full simulation is run for every agent to gather data about the KF performance.
2. Cost (Fitness) is calculated for each agent.
3. Based on Cost, agents are selected for reproduction.
4. New generation is created with crossover and mutation.
5. Repeat.

Crossover: each gene randomly selected from one of two parents, and has 5% chance to mutate.
Mutation: selected value from Gaussian $N(0, 0.01)$ is added to the gene.

Cost Function

Preface: Genome parameters directly affect the KF (localization), so evaluate using localization success.

Goal: Minimize the mean difference between the KF estimate and the ground truth position:

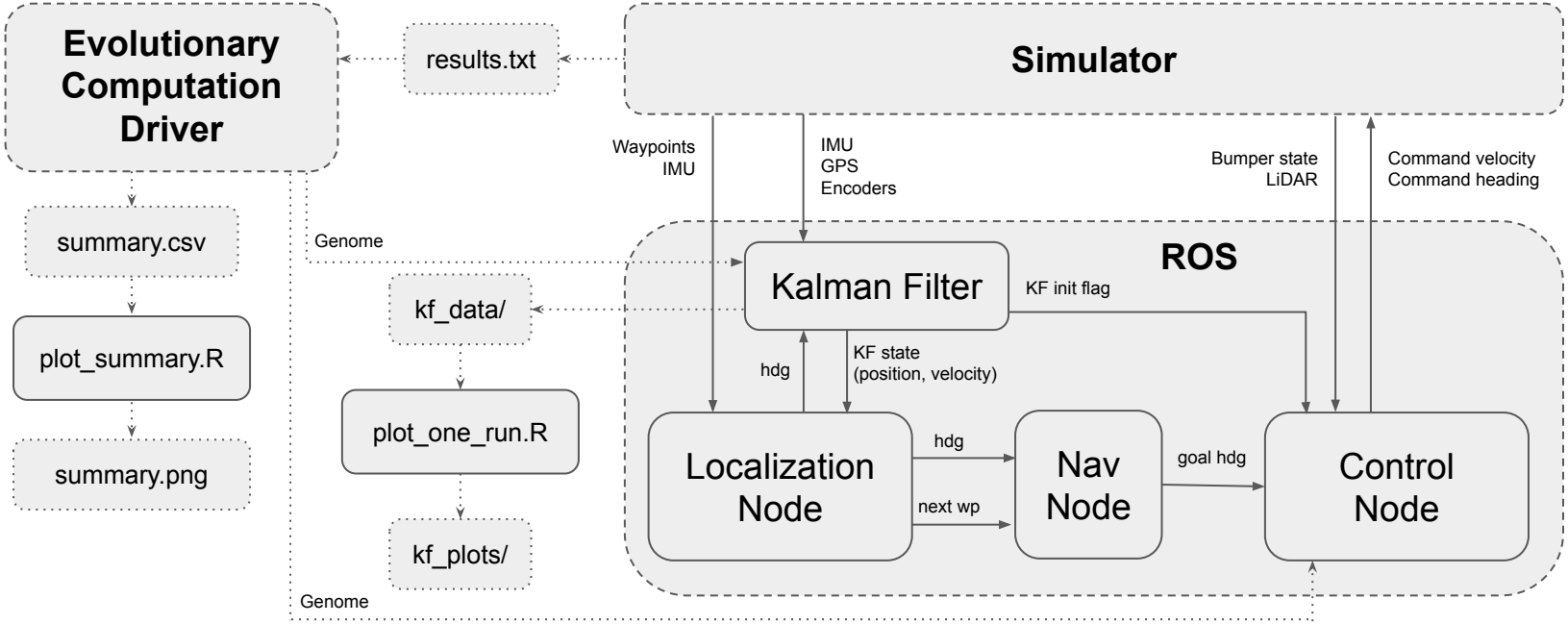
$$C = \frac{1}{N} \sum_{i=0}^{N} \left(|x_{i,state} - x_{i,true}| + |y_{i,state} - y_{i,true}| \right)$$

This cost, therefore, is a direct way of measuring the efficacy of our Kalman Filter.

Note: We can compute the average cost of using the raw sensor values without the KF.

- GPS has precision of ±5m for both longitude & latitude.
- Cost is then (5 + 5) = 10m.
- This is our benchmark performance.

Architecture with Machine Learning Included



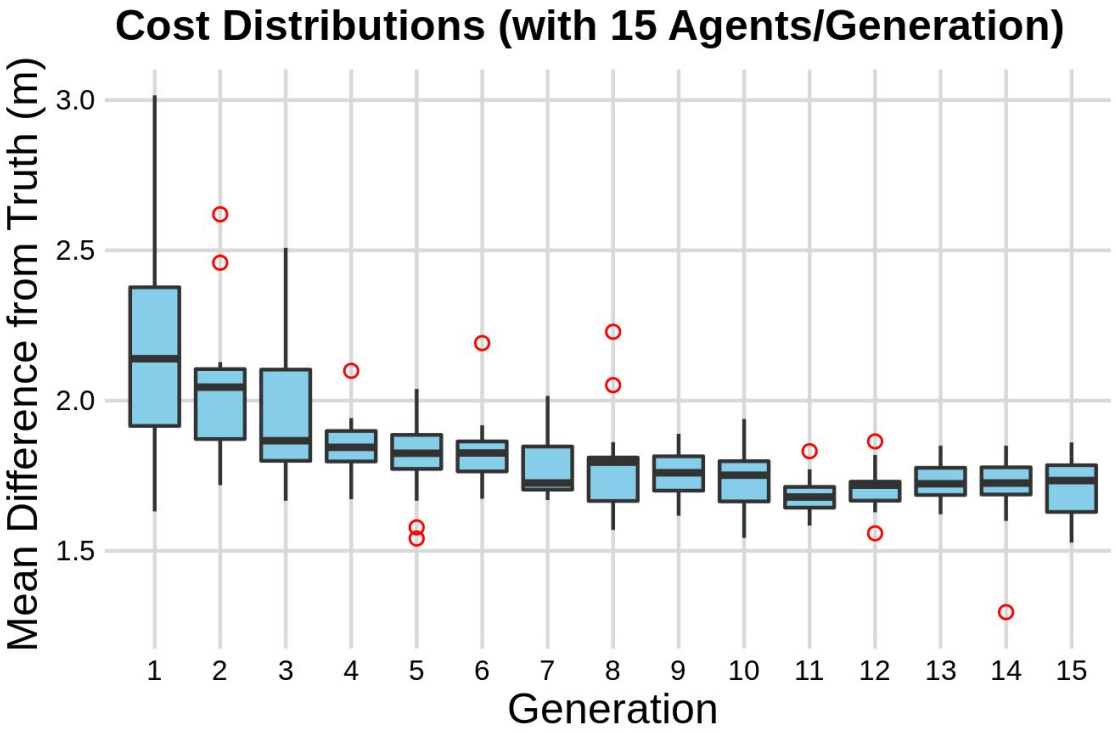
EC Effect

When instantiated with a random-valued genome, there is noticeable improvement in both the mean cost and the variance within each generation.

Over several sets of runs, the majority level off around 1.75m.

- Same optimum achieved by manual tuning.

Recall our benchmark cost was 10m, so this is over 5x better!



Evolution when initialized with randomized genome.

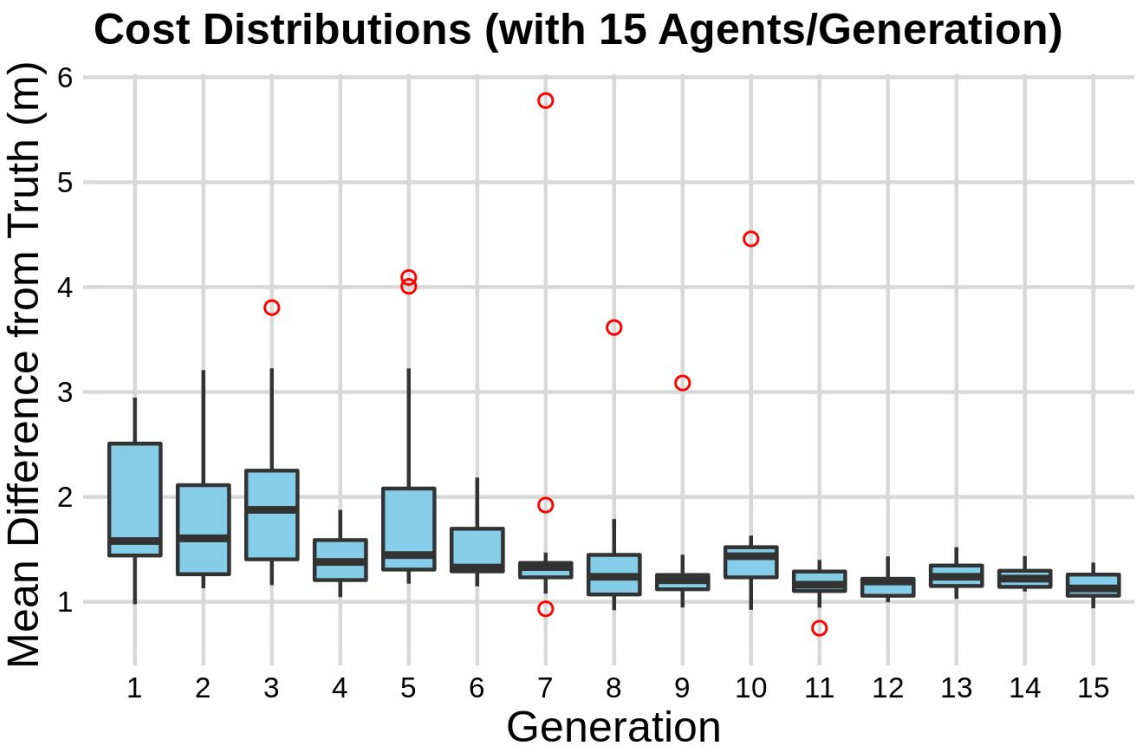
EC Effect

When the genome is instantiated with the values found from our previous manual tweaking, we see further improvements from the evolutionary tuning process.

- Levels off around 1.2m, improving the 1.75m from either the manual KF or the random evolution alone.

Variance decreases substantially, increasing consistency but hindering exploration.

- There could be a better optimum mean cost.



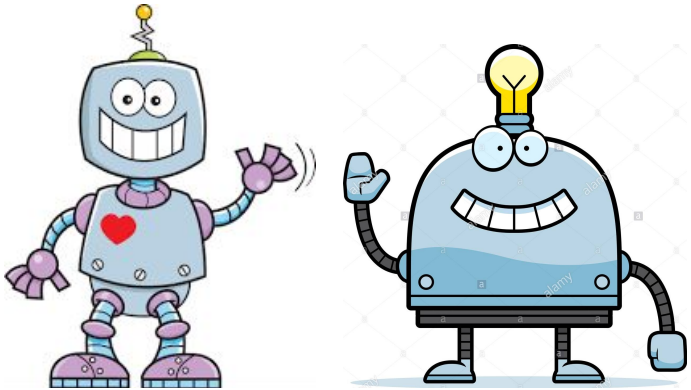
Evolution when initialized with a pre-tuned genome.

Conclusions

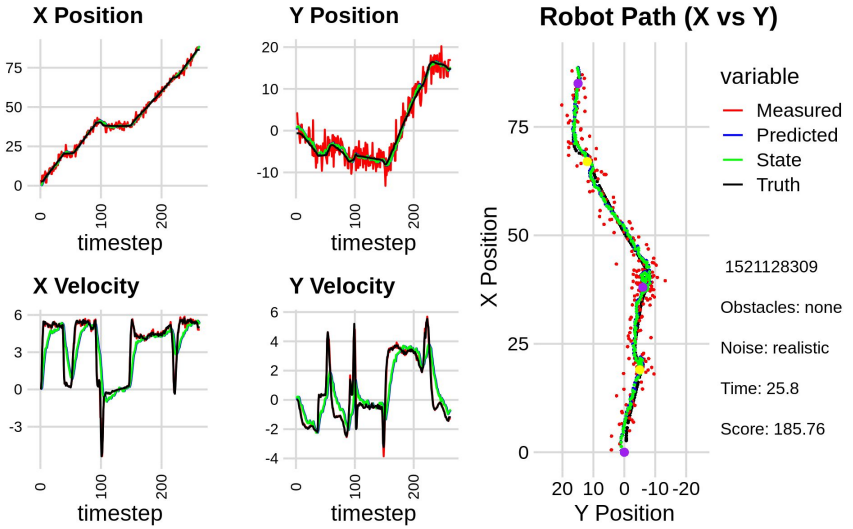
Localization was improved by the Kalman Filter with manual tuning.

Localization was equally improved by the Kalman Filter with automatic tuning.

Evolution after manual tuning achieves the best performance.



Images courtesy of VectorStock and Alamy Stock Photo.



Kalman Filter data from an agent after evolutionary training.

Future Work

I was not able to explore as wide an area as I'd hoped, and below are some of my untouched areas of interest. I invite others to continue this project by forking my GitHub repository: github.com/kevin-robb/capstone-kf-ml

- Attempt evolving motion parameters alone, rather than alongside localization.
- Attempt evolving a better motion policy for obstacle avoidance, possibly using a neural network.
- Experiment with including the LiDAR data in the KF's inputs to use obstacles and landmark detection for increased localization potential.
- Test this premise with a physical robot, probably requiring designing a simulator to match the real scenario as closely as possible.
- Work on correlating success or failure with certain genes to reduce the number of generations and agents necessary to converge on the optimum.

References

(Thanks to Dr. Hougen and the REAL Lab!)

Kalman Filter Tutorial, Alex Becker. Last modified 2018, www.kalmanfilter.net.

Michael Nielson, *Neural Networks and Deep Learning* (2019), Chapter 1, www.neuralnetworksanddeeplearning.com.

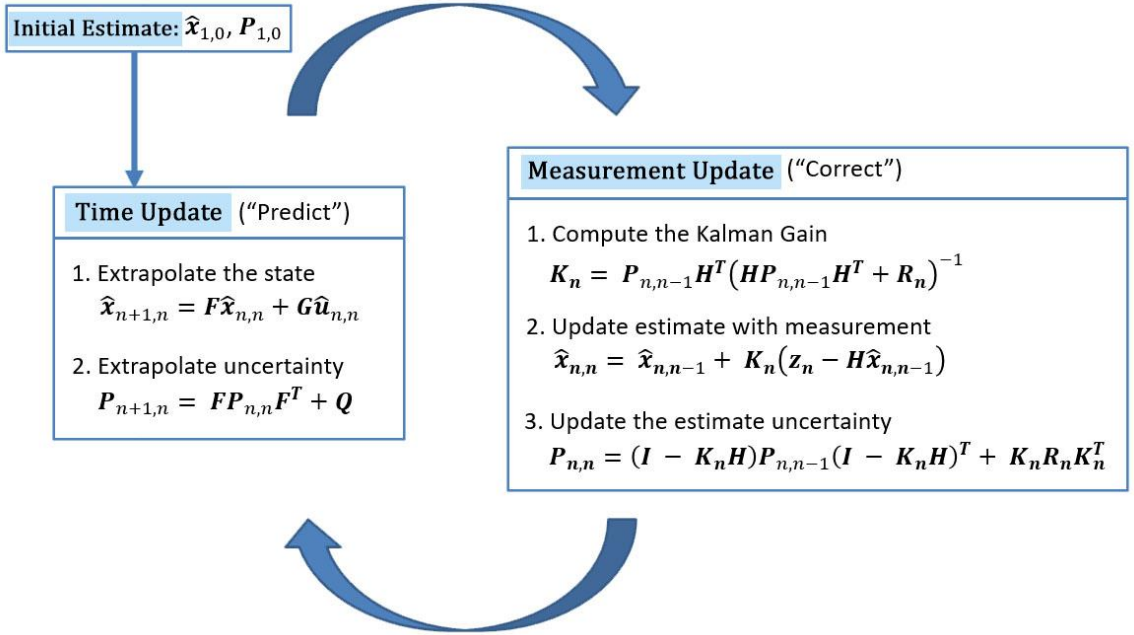
Jianguo Jack Wang, Weidong Ding, and Jinling Wang, "Improving Adaptive Kalman Filter in GPS/SDINS Integration with Neural Network," School of Surveying and Spatial Information Systems, University of New South Wales, Australia. *ResearchGate*, 2013, www.researchgate.net/publication/251439107_Improving_Adaptive_Kalman_Filter_in_GPSSDINS_Integration_with_Neural_Network

Justin Kleiber, *IGVC Kalman Filter Derivation*, Sooner Competitive Robotics, Intelligent Ground Vehicle Competition design document submission, March 16th 2020.

Noah Zemlin, *SCR Software Challenge Simulator*, Sooner Competitive Robotics, 2020, github.com/SoonerRobotics/SCR-SWC-20

Appendix: The Kalman Filter Details

Many of these equations reduce in our application due to our setup.



Multidimensional Kalman Filter equation cycle. Modified figure from kalmanfilter.net to highlight most important focus.

Appendix: State and State Transition Matrix

We use the x and y positions and velocities for our 4D State, ***X*** (in global coordinates).

$$X = \begin{pmatrix} x & y & \dot{x} & \dot{y} \end{pmatrix}^T$$

The State Transition Matrix, ***F***, is used to predict the next state. Matrix multiplication with the State applies the system of equations shown. This is our constant velocity model.

$$\boldsymbol{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{cases} \hat{x}_{n+1,1} = \hat{x}_{n,n} + \hat{\dot{x}}_{n,n} \cdot \Delta t \\ \hat{y}_{n+1,1} = \hat{y}_{n,n} + \hat{\dot{y}}_{n,n} \cdot \Delta t \\ \hat{\dot{x}}_{n+1,1} = \hat{\dot{x}}_{n,n} \\ \hat{\dot{y}}_{n+1,1} = \hat{\dot{y}}_{n,n} \end{cases}$$

Appendix: Measurements and Observation Matrix

Our measurements are GPS coordinates (λ, Λ), linear velocity (*v*), and the current global heading (***ϕ***). We obtain these from our sensor suite, which includes GPS, IMU, and motor encoders. We transform these to a 4D vector, ***Z***:

$$Z = \begin{bmatrix} \lambda \\ \Lambda \\ v_x \\ v_y \end{bmatrix} = \begin{bmatrix} (\lambda - \lambda_{start}) \cdot \text{lat\_to\_m} \\ (\Lambda - \Lambda_{start}) \cdot \text{lon\_to\_m} \\ v \cdot \cos\phi \\ -v \cdot \sin\phi \end{bmatrix}$$

The Observation Matrix, ***H***, relates the actual measurements, ***Z***, to our state. Due to our choice of state variables and our transformations to ***Z***, ***H*** is simply the identity matrix. This lets us simplify many of the KF equations.

$$\boldsymbol{H} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \boldsymbol{I}$$

Appendix: Covariance (Estimate Uncertainty)

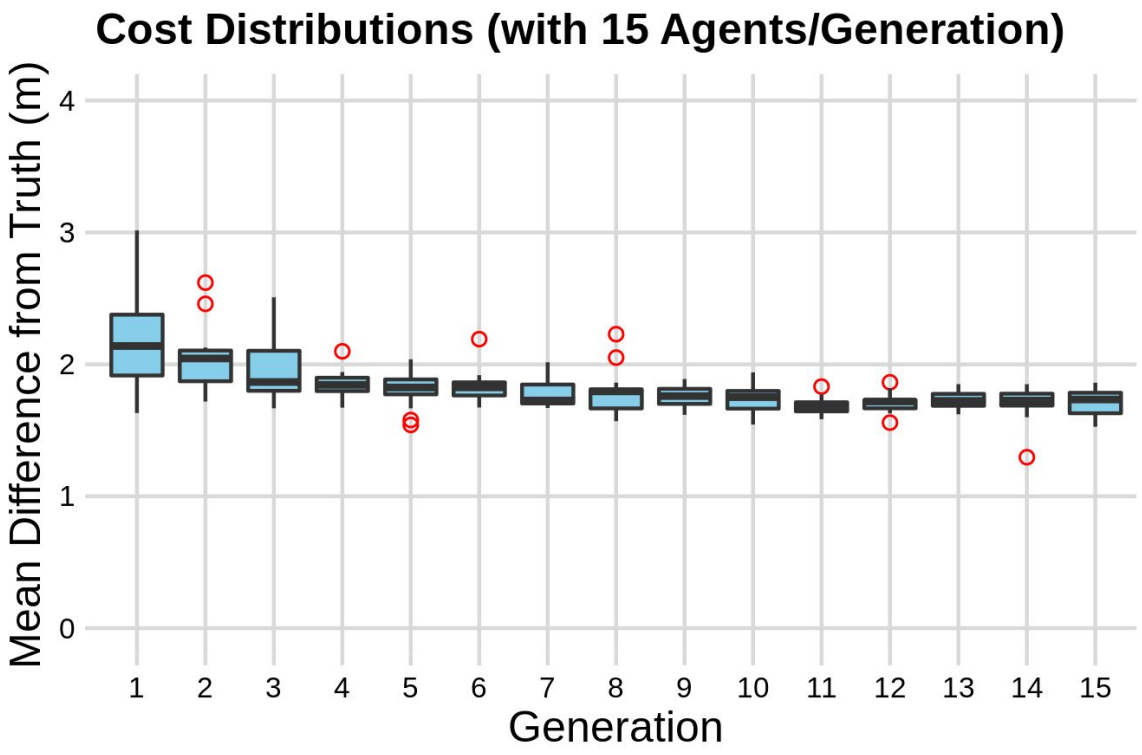
The Covariance Matrix, ***P***, tells us the uncertainty in the KF's own estimates. We initialize it with the variances of our measurements, and tune it as the filter runs.

$$\boldsymbol{P} = \begin{pmatrix} \sigma_x^2 & 0 & 0 & 0 \\ 0 & \sigma_y^2 & 0 & 0 \\ 0 & 0 & \sigma_{v_x}^2 & 0 \\ 0 & 0 & 0 & \sigma_{v_y}^2 \end{pmatrix} = \begin{pmatrix} 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0 & 0 & 0.25 & 0 \\ 0 & 0 & 0 & 0.5 \end{pmatrix}$$

Increases during Time Update and decreases in Measurement Update, so there's a net increase if we don't get confirmation that the estimates are good.

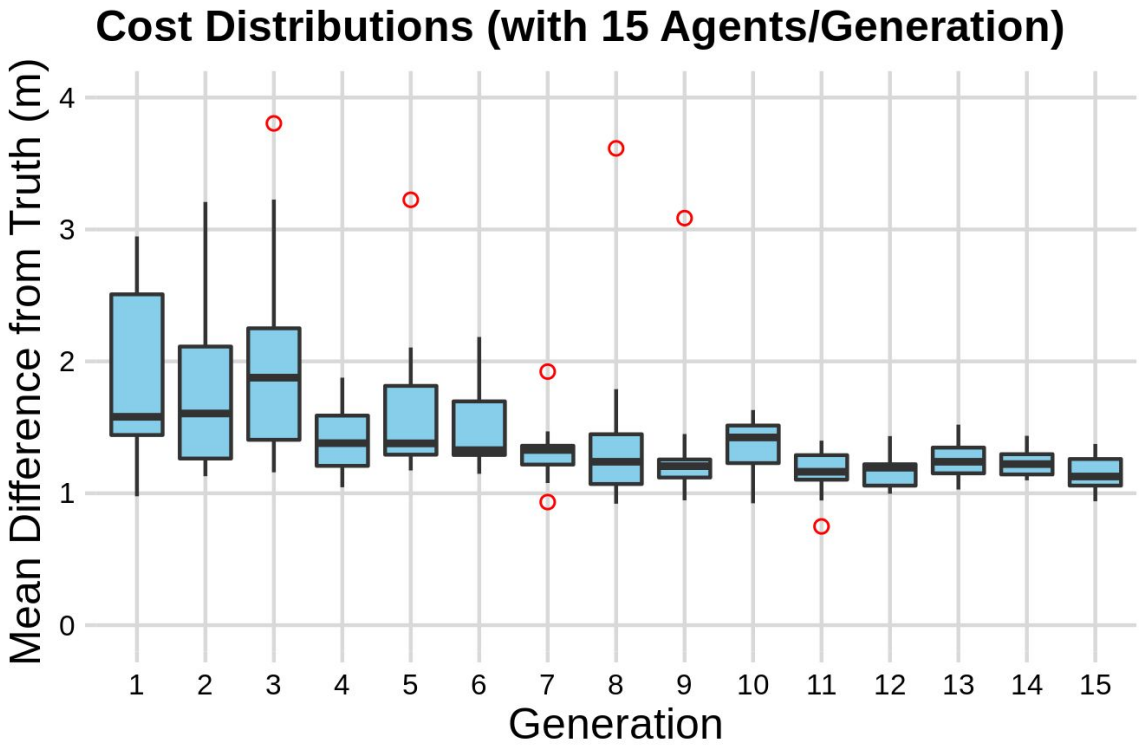
Process Noise, ***Q***, and Measurement Uncertainty, ***R*** are hard to calculate exactly.
Approximate as ***Q*** = ***R*** = 0.01 ***I*** for now.

Appendix: EC Graphs to Scale



Evolution when initialized with randomized genome.

Appendix: EC Graphs to Scale



Evolution when initialized with a pre-tuned genome.