Parallel Particle Filter in Julia

Gustavo Goretkin

December 12, 2011

Introduction

First a disclaimer

- This project is not finished.

The project in a sentence.

- Implement a particle filter in Julia that takes advantage of distributed-memory parallelism.

Particle Filter

- An approximation to the general Bayes filter

Particle Filter

- An approximation to the general Bayes filter
- Track the state of a dynamical system

Particle Filter

- An approximation to the general Bayes filter
- Track the state of a dynamical system
    - but the state is not directly observable
    - but the dynamical system is noisy

Particle Filter

- An approximation to the general Bayes filter
- Track the state of a dynamical system
  - but the state is not directly observable
  - but the dynamical system is noisy
- Same concept as the Kalman filter, but fewer assumptions

Particle Filter

- An approximation to the general Bayes filter
- Track the state of a dynamical system
  - but the state is not directly observable
  - but the dynamical system is noisy
- Same concept as the Kalman filter, but fewer assumptions
  - but the system dynamics may be non-linear
  - the observation function may be non-linear
  - the process noise and and observation noise may be non-Gaussian
  - the hypothesis is not confined to be Gaussian – can have multimodal hypotheses

Particle Filter workings

1. Start with a set of n particles at step t_{n-1}. These particles represent the hypothesis at that time.

Particle Filter workings

1. Start with a set of n particles at step t_{n-1}. These particles represent the hypothesis at that time.
2. Propagate each particle independently according to system dynamics. This is the *a priori* hypothesis.

Particle Filter workings

1. Start with a set of n particles at step t_{n-1}. These particles represent the hypothesis at that time.
2. Propagate each particle independently according to system dynamics. This is the *a priori* hypothesis.
3. Make an observation and weight each particle by the likelihood of the the particle – $p\left(\text{observation}\middle|\text{particle}\right)$

Particle Filter workings

1. Start with a set of n particles at step t_{n-1}. These particles represent the hypothesis at that time.
2. Propagate each particle independently according to system dynamics. This is the *a priori* hypothesis.
3. Make an observation and weight each particle by the likelihood of the the particle – $p\left(\text{observation}\middle|\text{particle}\right)$
4. Resample n particles according to their weights. This represents the *a posteriori* hypothesis at t_n.

Particle Filter workings

1. Start with a set of n particles at step t_{n-1}. These particles represent the hypothesis at that time.
2. Propagate each particle independently according to system dynamics. This is the *a priori* hypothesis.
3. Make an observation and weight each particle by the likelihood of the the particle – $p\left(\text{observation}|\text{particle}\right)$
4. Resample n particles according to their weights. This represents the *a posteriori* hypothesis at t_n.

   (a) many different resampling techniques with different computation complexities and variances.

Particle Filter workings

1. Start with a set of n particles at step t_{n-1}. These particles represent the hypothesis at that time.
2. Propagate each particle independently according to system dynamics. This is the *a priori* hypothesis.
3. Make an observation and weight each particle by the likelihood of the the particle – $p\left(\text{observation}\middle|\text{particle}\right)$
4. Resample n particles according to their weights. This represents the *a posteriori* hypothesis at t_n.

   (a) many different resampling techniques with different computation complexities and variances.
   (b) the part that is not embarrassingly parallel.

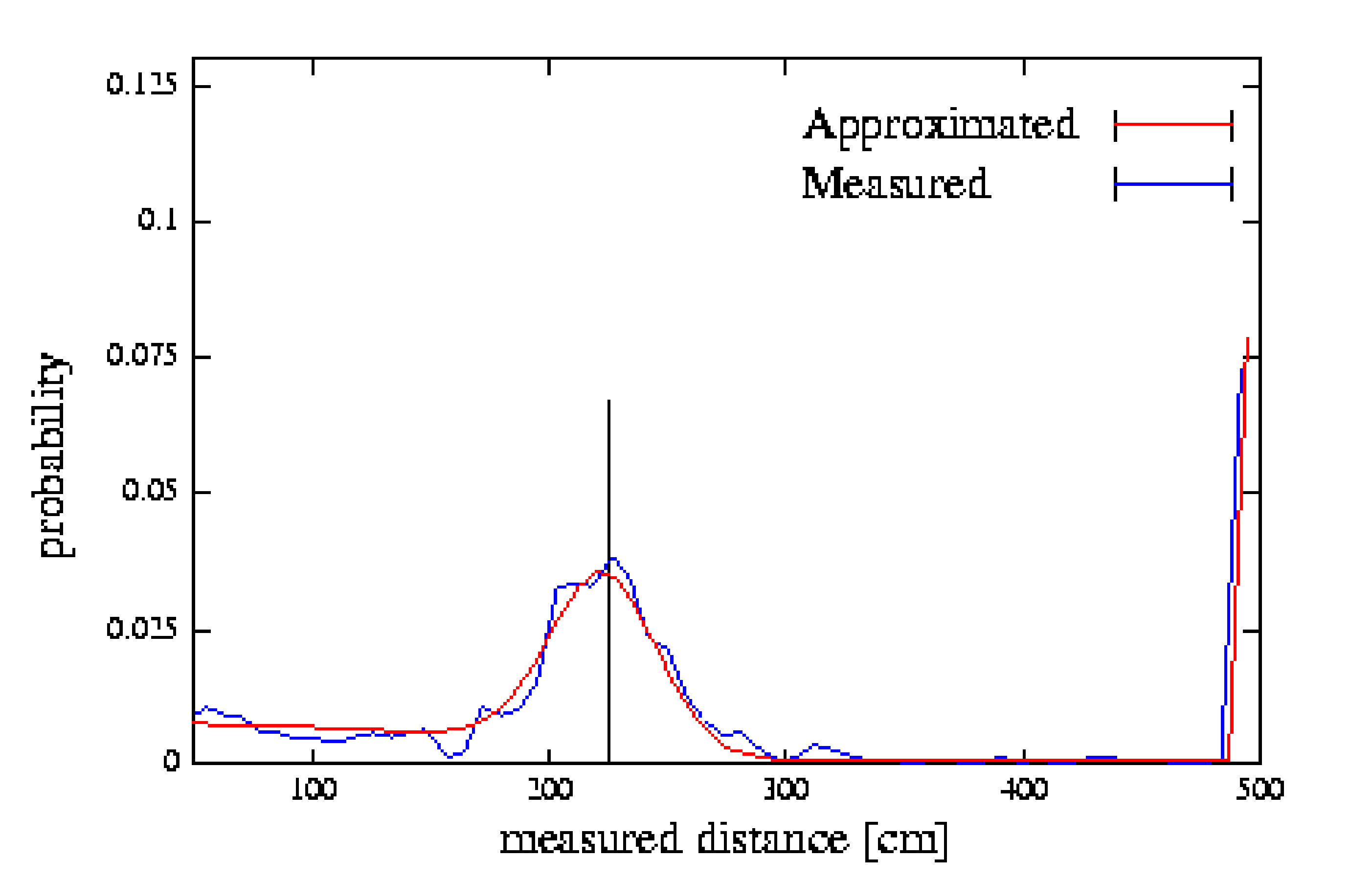
Particle Filter workings

1. Start with a set of n particles at step t_{n-1}. These particles represent the hypothesis at that time.
2. Propagate each particle independently according to system dynamics. This is the *a priori* hypothesis.
3. Make an observation and weight each particle by the likelihood of the the particle – $p\left(\text{observation}|\text{particle}\right)$
4. Resample n particles according to their weights. This represents the *a posteriori* hypothesis at t_n.

   (a) many different resampling techniques with different computation complexities and variances.
   (b) the part that is not embarrassingly parallel.

Robot Localization Application

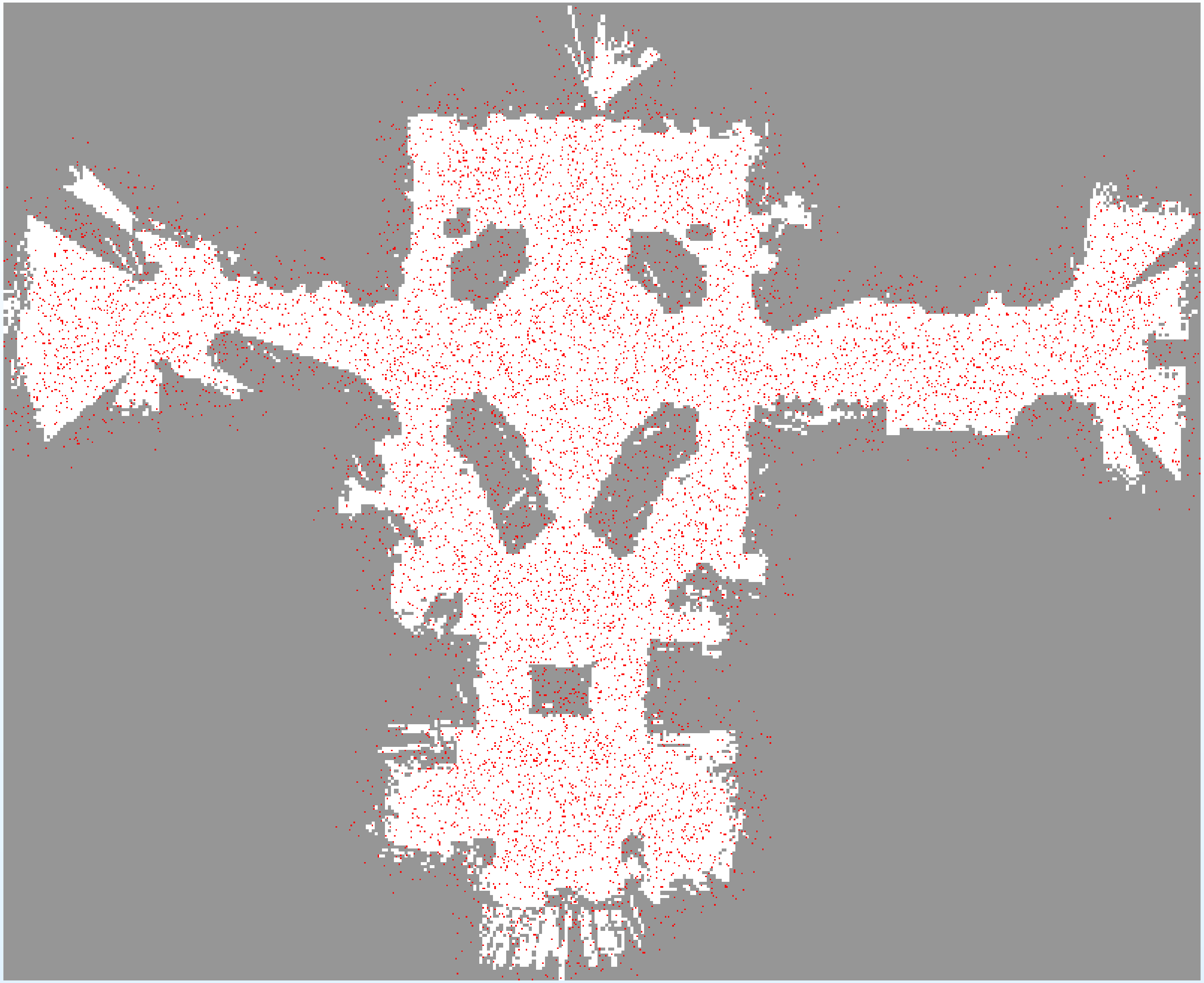
Robot Localization Application



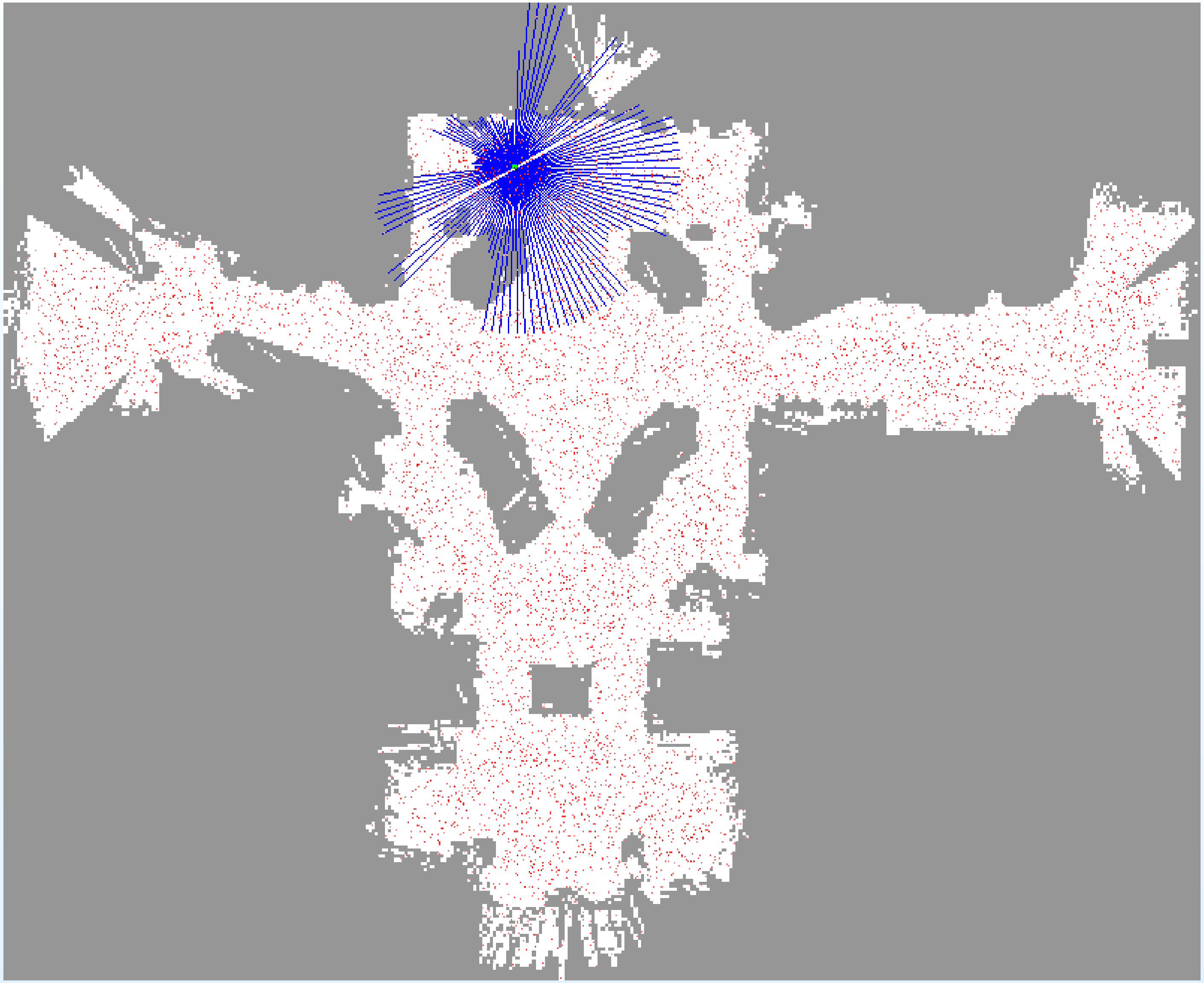
1

[1] Taken from Probabilistic Robotics 2005

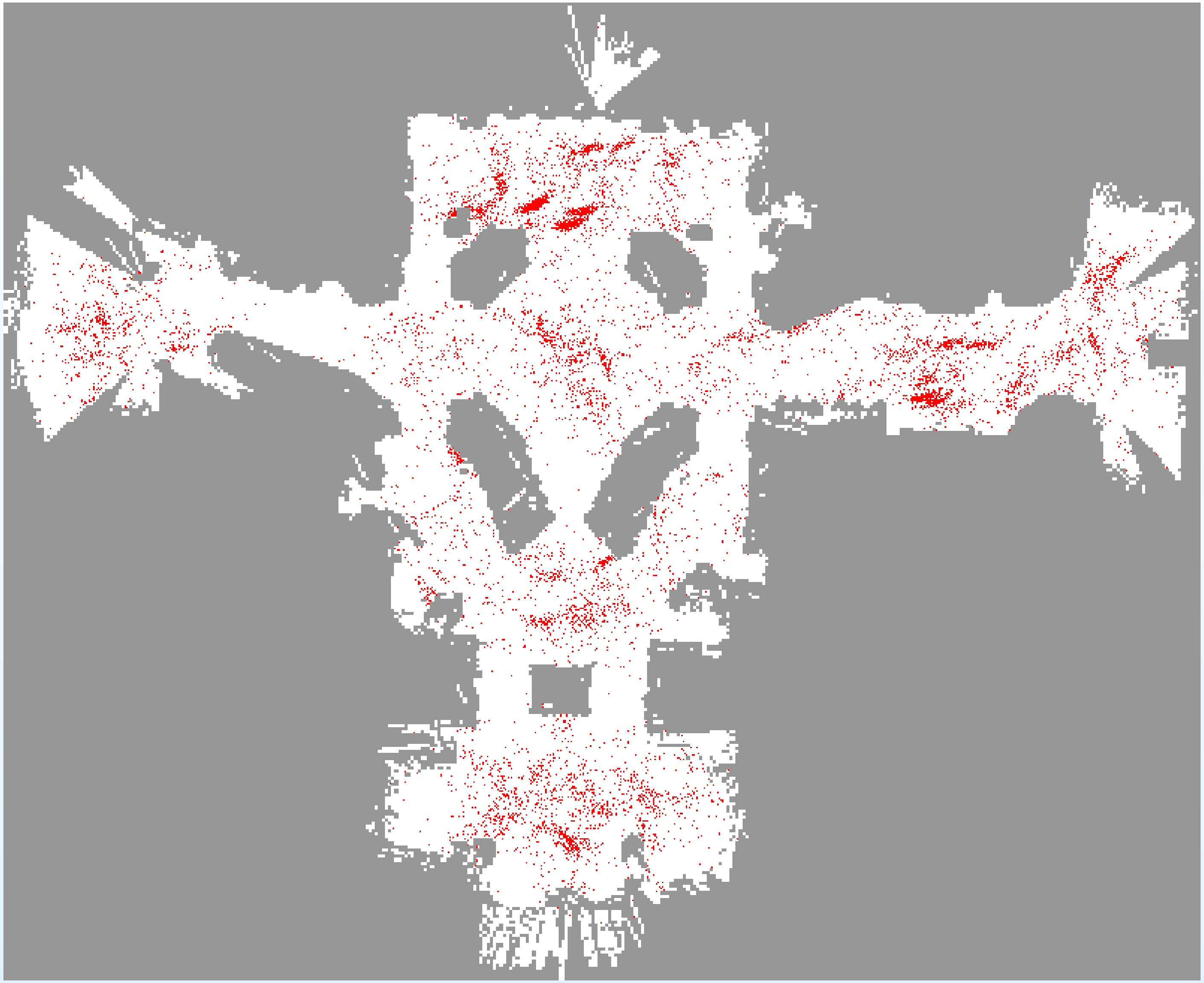
Robot Localization Application



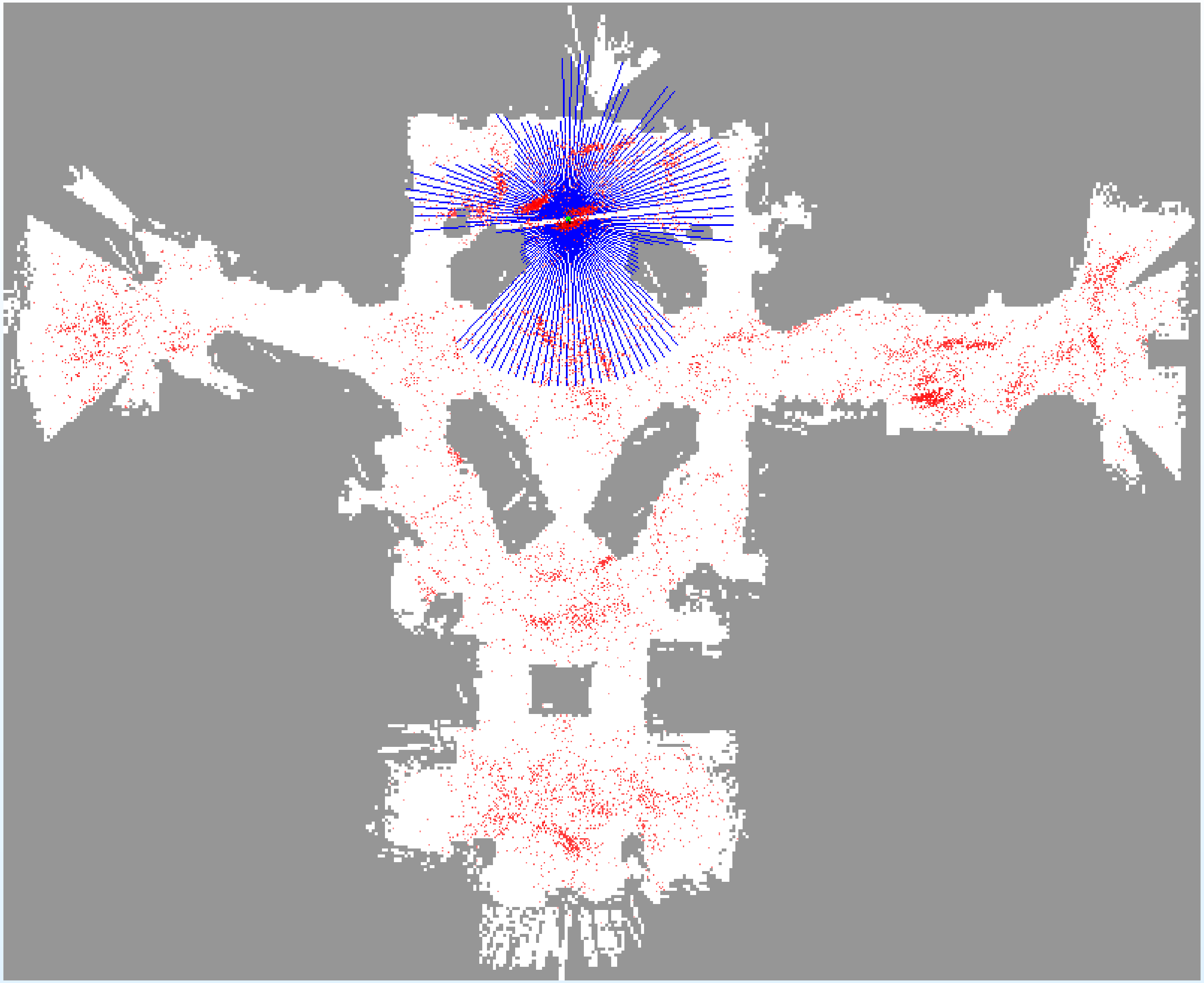
Robot Localization Application



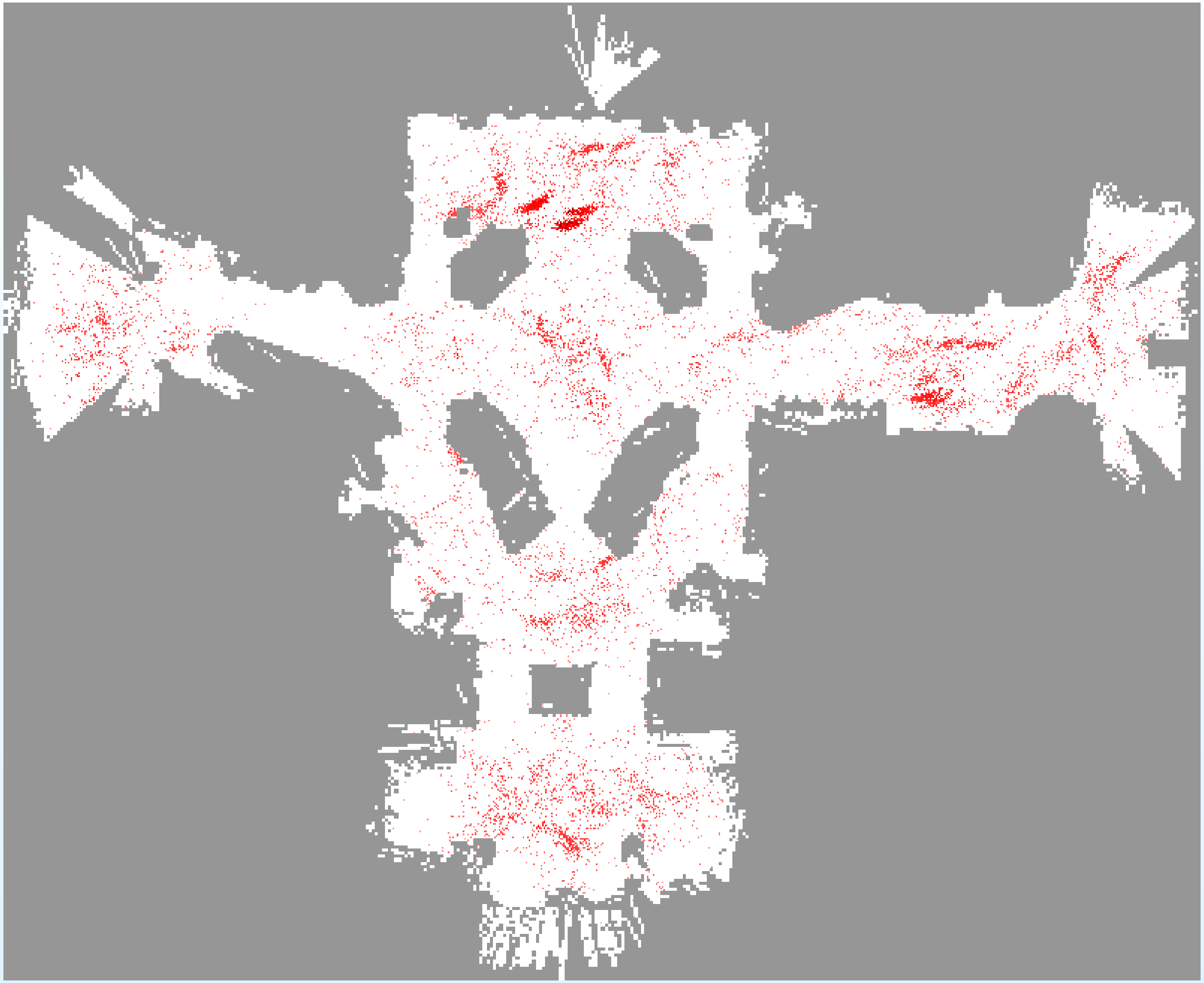
Robot Localization Application



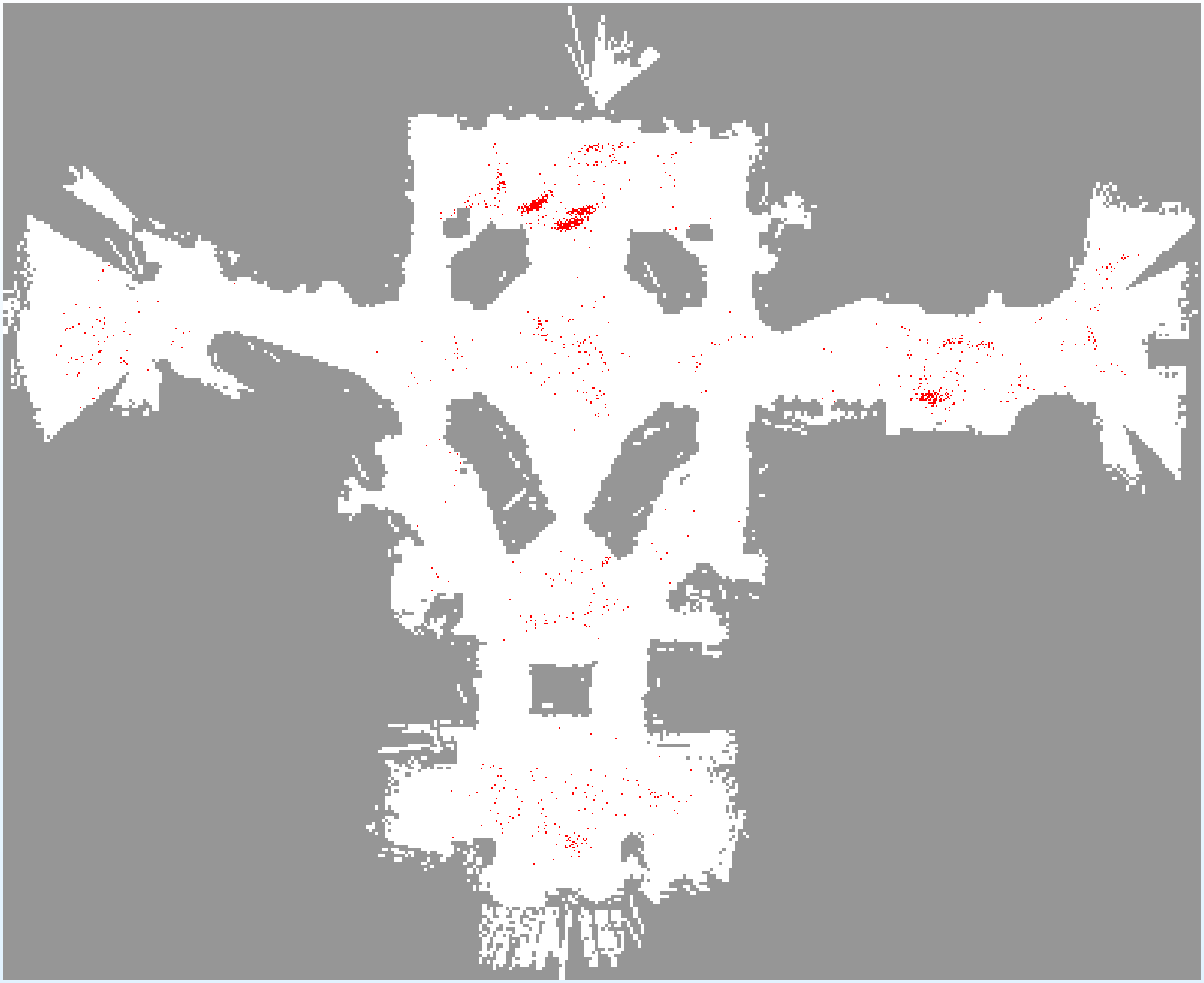
Robot Localization Application



Robot Localization Application

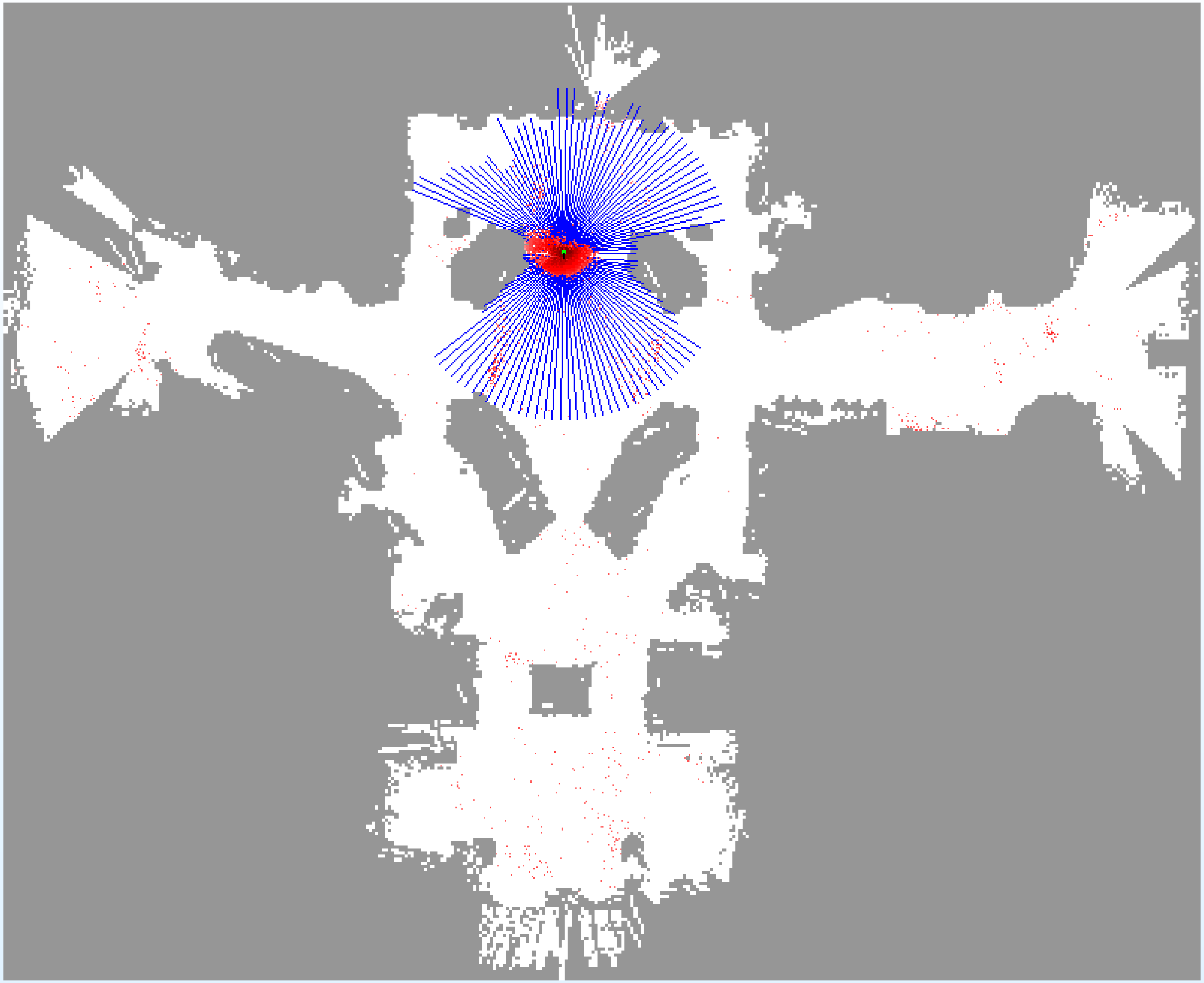


Robot Localization Application

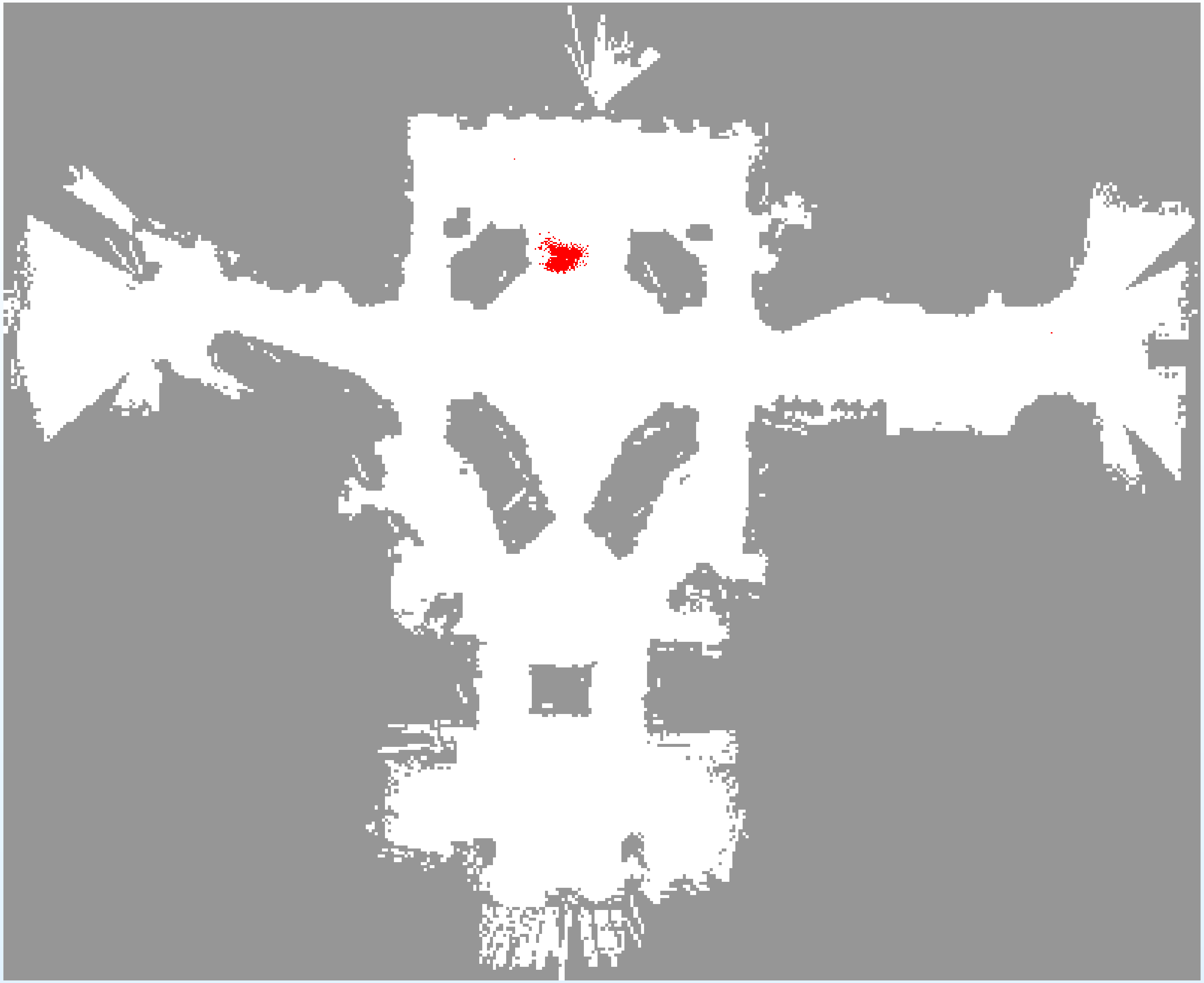


Robot Localization Application

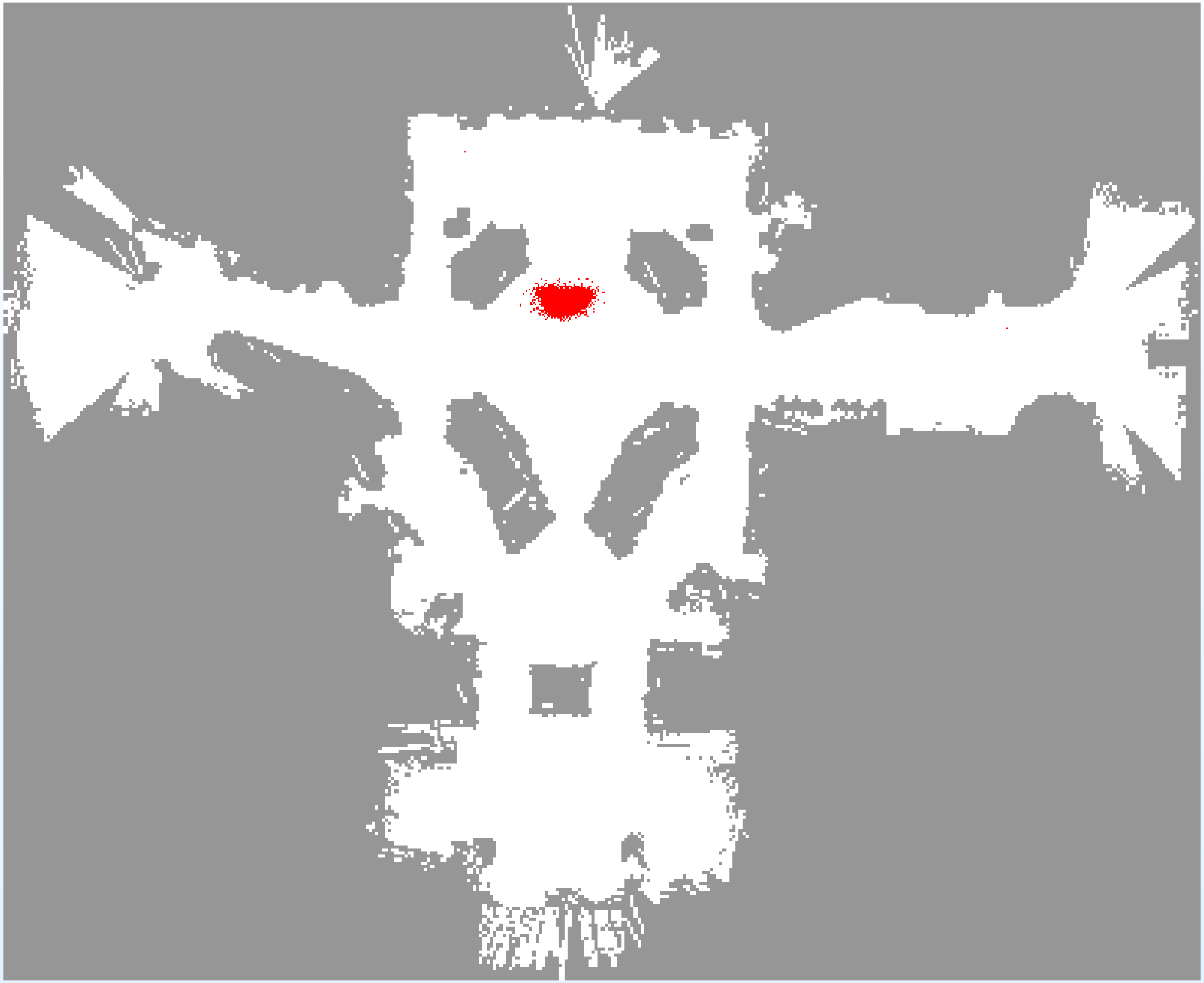
Robot Localization Application



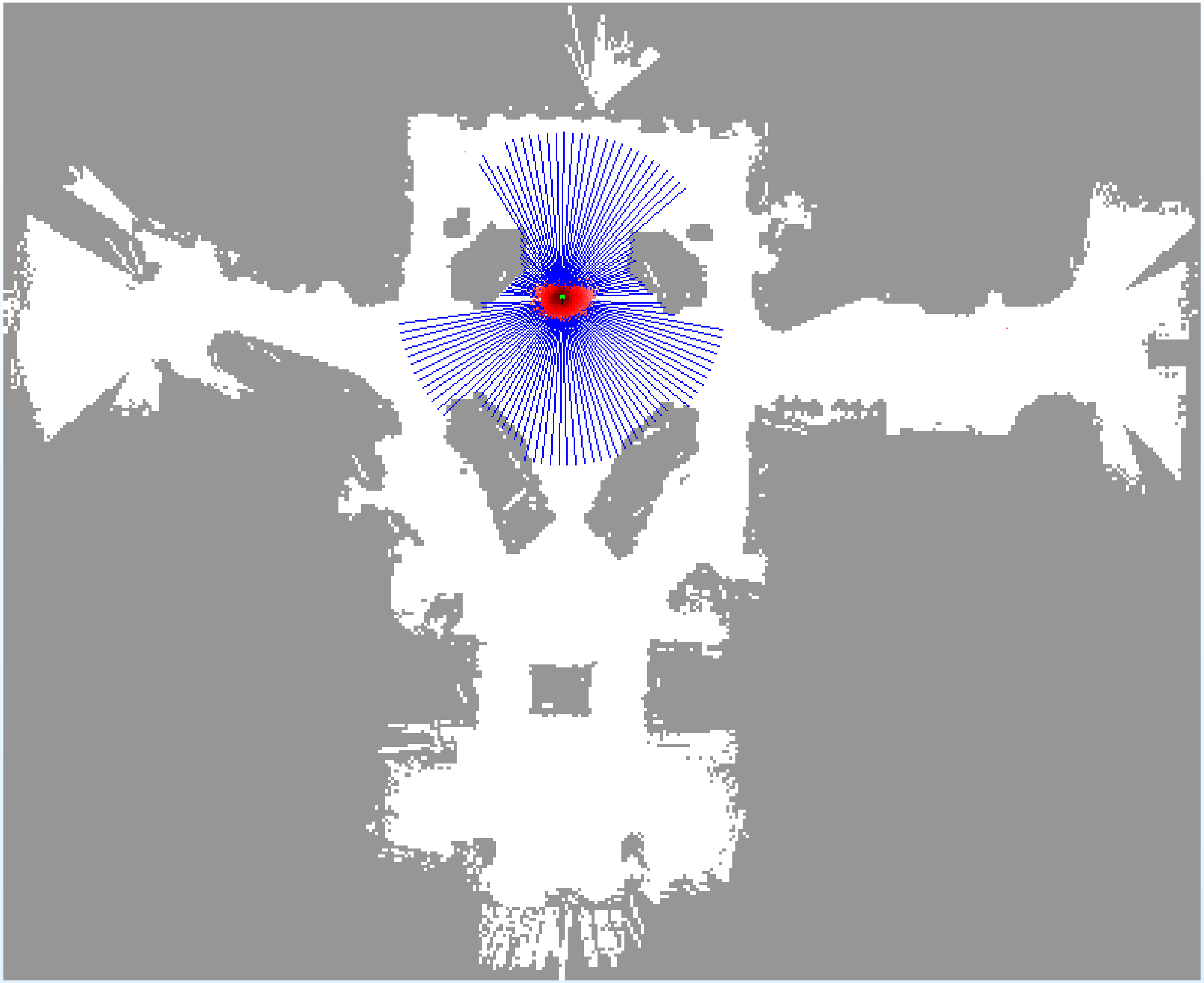
Robot Localization Application



Robot Localization Application

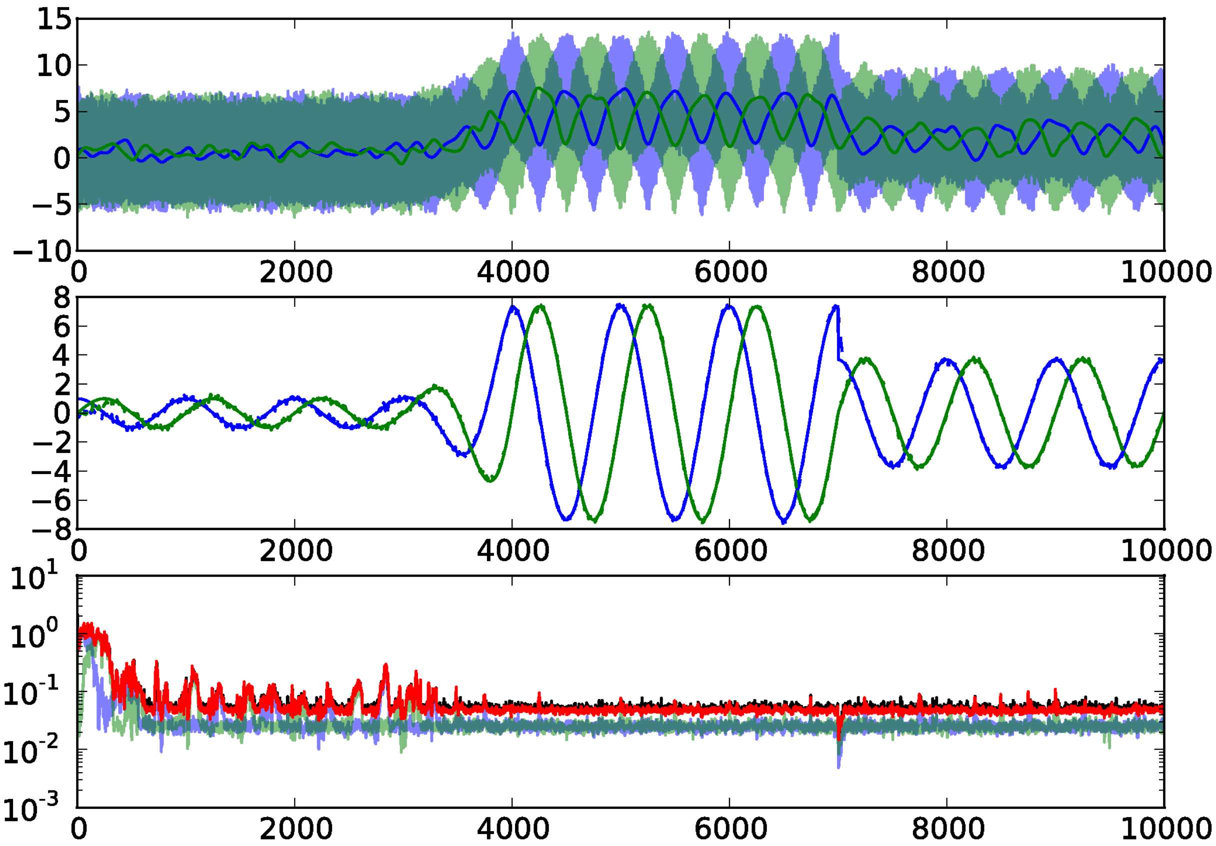


Robot Localization Application

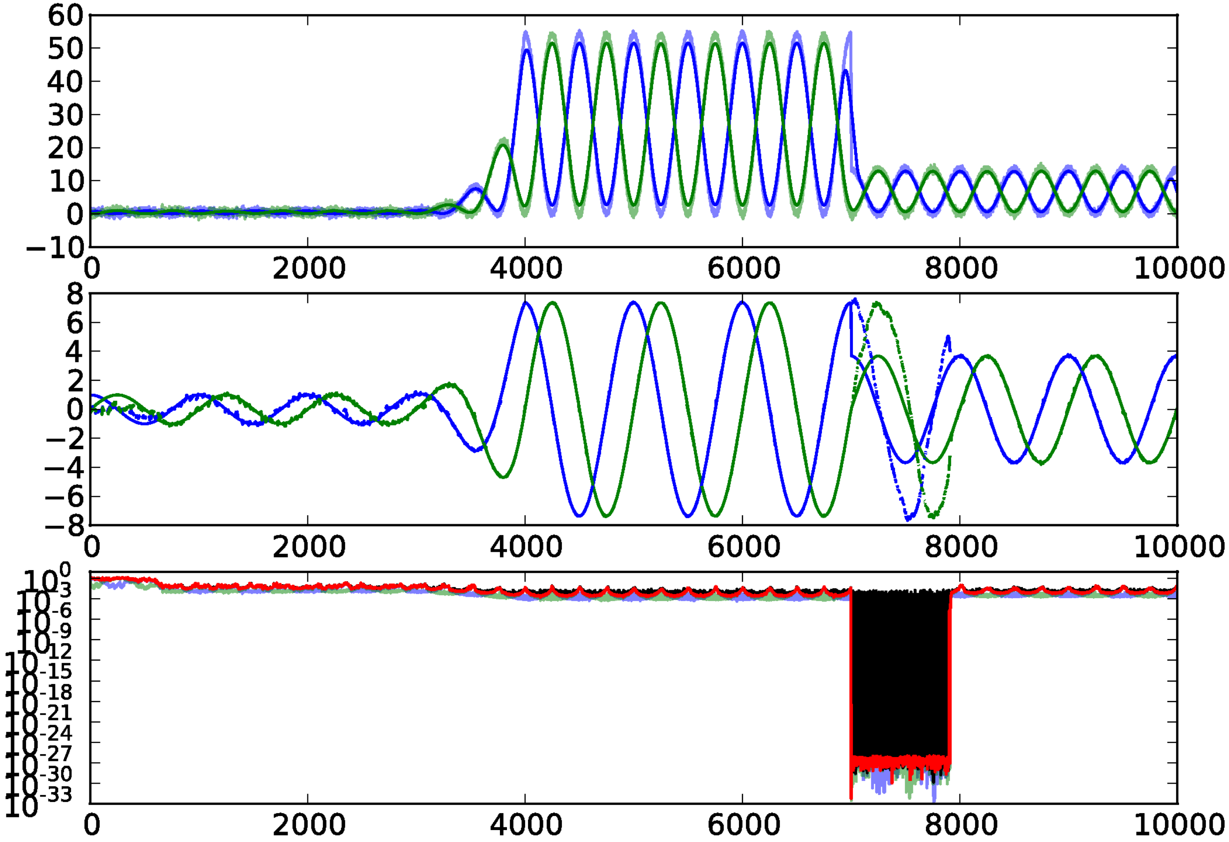


Simple Example

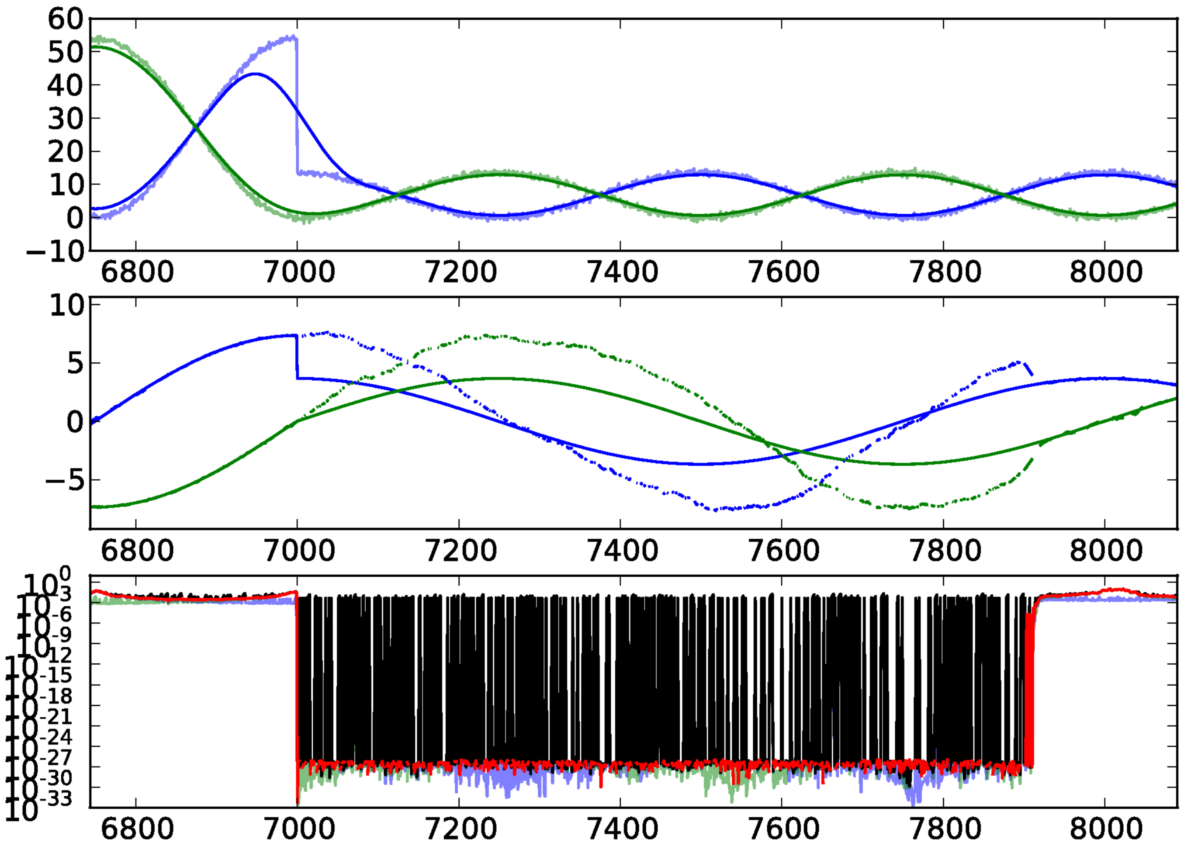
Absolute Value observation $n = 1000$



Square Value observation $n = 500$



Square Value observation $n = 500$



Project

Goal of the project

- Implement a practical parallel particle filter for distributed memory systems

Goal of the project

- Implement a practical parallel particle filter for distributed memory systems
  - Bayesian Filtering libraries exist

Goal of the project

- Implement a practical parallel particle filter for distributed memory systems
    - Bayesian Filtering libraries exist
    - I didn't find any for distributed memory systems.

Goal of the project

- Implement a practical parallel particle filter for distributed memory systems
    - Bayesian Filtering libraries exist
    - I didn't find any for distributed memory systems.
        - Maybe for a good reason...

Goal of the project

- Implement a practical parallel particle filter for distributed memory systems
  - Bayesian Filtering libraries exist
  - I didn't find any for distributed memory systems.
    - Maybe for a good reason...
- Create a framework general enough so that different parameters of the particle filter can be tested
  - plugging in different sampling techniques, noise parameters, etc.

Goal of the project

- Implement a practical parallel particle filter for distributed memory systems
  - Bayesian Filtering libraries exist
  - I didn’t find any for distributed memory systems.
    - Maybe for a good reason...
- Create a framework general enough so that different parameters of the particle filter can be tested
  - plugging in different sampling techniques, noise parameters, etc.
  - benchmark computational performance and estimation performance

Goal of the project

- Implement a practical parallel particle filter for distributed memory systems
  - Bayesian Filtering libraries exist
  - I didn't find any for distributed memory systems.
    - Maybe for a good reason...
- Create a framework general enough so that different parameters of the particle filter can be tested
  - plugging in different sampling techniques, noise parameters, etc.
  - benchmark computational performance and estimation performance
- Provide a use case of Julia. Hopefully create an elegant implementation that might attract others to the platform
  - especially think about Monte Carlo Bayesian inference folks

Current State of the Project

- Implementation still pending. I'm still learning Julia.

Current State of the Project

- Implementation still pending. I'm still learning Julia.
- SIMD approach.
  - Each compute node starts with a balanced set of particles.
  - The state propagation and observation functions happen independently on each machine.
  - After an independent resampling step, particles will likely be unbalanced.

Current State of the Project

- Implementation still pending. I'm still learning Julia.
- SIMD approach.
  - Each compute node starts with a balanced set of particles.
  - The state propagation and observation functions happen independently on each machine.
  - After an independent resampling step, particles will likely be unbalanced.
    - Balancing between the two largest offenders

Current State of the Project

- Implementation still pending. I'm still learning Julia.
- SIMD approach.
  - Each compute node starts with a balanced set of particles.
  - The state propagation and observation functions happen independently on each machine.
  - After an independent resampling step, particles will likely be unbalanced.
    - Balancing between the two largest offenders
- DArray primitives implemented
  - argmin, argmax, cumsum

Current State of the Project

- Implementation still pending. I'm still learning Julia.
- SIMD approach.
  - Each compute node starts with a balanced set of particles.
  - The state propagation and observation functions happen independently on each machine.
  - After an independent resampling step, particles will likely be unbalanced.
    - Balancing between the two largest offenders
- DArray primitives implemented
  - argmin, argmax, cumsum
- Other primitives
  - binary search, independent sampling, redistribution

Current State of the Project

- Implementation still pending. I'm still learning Julia.
- SIMD approach.
  - Each compute node starts with a balanced set of particles.
  - The state propagation and observation functions happen independently on each machine.
  - After an independent resampling step, particles will likely be unbalanced.
    - Balancing between the two largest offenders
- DArray primitives implemented
  - argmin, argmax, cumsum
- Other primitives
  - binary search, independent sampling, redistribution

Future Work

- Consider other rebalancing approaches
  - solving a minimum-cost graph flow problem to rebalance particles

Future Work

- Consider other rebalancing approaches
  - solving a minimum-cost graph flow problem to rebalance particles
- Think about memory allocation – minimize memory allocation per iteration.

Future Work

- Consider other rebalancing approaches
  - solving a minimum-cost graph flow problem to rebalance particles
- Think about memory allocation – minimize memory allocation per iteration.
- Think differently about parallelism. I'm stuck in the MPI mentality – probably not exploiting DArrays enough.

Future Work

- Consider other rebalancing approaches
  - solving a minimum-cost graph flow problem to rebalance particles
- Think about memory allocation – minimize memory allocation per iteration.
- Think differently about parallelism. I'm stuck in the MPI mentality – probably not exploiting DArrays enough.