PILOT: An Actor-oriented Learning and Optimization Toolkit for Robotic Swarm Applications *

Ilge Akkaya
UC Berkeley
ilgea@eecs.berkeley.edu

Shuhei Emoto
IHI Corporation
syuuhei_emoto@ihi.co.jp

Edward A. Lee
UC Berkeley
eal@eecs.berkeley.edu

ABSTRACT

We present PILOT (Ptolemy Inference, Learning, and Optimization Toolkit), an actor-oriented machine learning and optimization toolkit that is designed for developing data intensive distributed applications for sensor networks. We define an actor interface that bridges state-space models for robotic control problems and a collection of machine learning and optimization algorithms, then demonstrate how the framework leverages programmability of sophisticated distributed robotic applications on streaming data. As a case study, we consider a cooperative target tracking scenario and study how the framework enables adaptation and implementation of control policies and simulation within environmental constraints by presenting actor-oriented abstractions that enable application developers to build state-space aware machine learning and optimization actors.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Design, Algorithms

Keywords

State-space modeling, system design, robotic swarms

1. INTRODUCTION

The rapid growth of networked smart sensors today offers unprecedented volumes of continually streaming data about the physical world. Making effective use of the data will require data-to-information technology that is qualitatively different from traditional data analytics. Distributed robotic applications is an important class of such data intensive applications that demand real-time guarantees.

In classical control systems, sensor data provide a direct measurement of a physical quantity to be controlled. In modern control systems, the data provided by sensors is more indirect. Consider, for example, a robot that is trying to track a target. It may use a combination of sensors, including cameras, laser or ultrasonic range-finders, signal-strength measurements, microphones, etc. None of these present data that directly measures a directly controllable quantity. These data must be converted to actionable information, e.g., a concise summary of the estimated location and target trajectory. Given such a concise summary, an optimal trade-off between often conflicting objectives must be found and translated into action. To be effective, the data summarization and optimization must be performed in real time on streaming data.

The complex nature of applications that involve sensing, data processing, control and actuation requires a structured system modeling workflow. Distributed system design has historically been prone to errors due to timing and concurrency requirements. In the cyber-physical setting a robotic swarm operates in, the problem persists, if not made worse.

This paper describes a software architecture for enabling an actor-oriented approach to developing distributed robotic sensing and control applications. The presented toolkit enables inference and optimization tasks to be defined by the state-space models of the problem domain. This level of abstraction presents analysis and optimization components that are reusable and less error-prone, and can be easily deployed in a distributed setting while preserving computational semantics.

A key goal of PILOT is to enable system engineers who are not experts in machine learning to use the toolkit in order to develop applications that rely on on-line estimation and inference. In this context, we adopt an actor-oriented design and provide domain-specific specializations of general machine learning techniques. Moreover, we leverage a design framework that boosts programmability of swarm applications by providing abstractions in which algorithmic accuracy can be traded off for better real-time performance under scarcity of resources.

The rest of the paper is organized as follows: In section 2, we survey previous research on actor-oriented design and recent results on cooperative robotic control applications and their computational implications. Section 3 summarizes the application requirements posed by the developments in robotic sensor networks, followed by Section 4, which details the proposed toolkit interface for state-space aware inference

*This work was supported in part by the TerraSwarm Research Center, one of six centers administered by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

and optimization workflow. Section 5 presents an application scenario on robot swarms that utilizes PILOT actors. Conclusions are presented in Section 6.

2. RELATED WORK

Although individual efforts for control policies and swarm control exist, programming complex control and inference algorithms for sensor networks still remains a challenge, due to the lack of a unifying framework that provides interfaces that are reusable, extendable, and easy to apply to streaming data. Specialized platforms that leverage development of pattern recognition tasks on interactive devices have been proposed [6, 10]. One other related framework in this area is presented in [1], which introduces a high-level programming language for robotic applications designed on ROS [14].

The multidisciplinary field of CPS design has triggered intensive research on programming foundations of distributed heterogeneous systems [3, 8]. Researchers have investigated a collection of programming paradigms to be able to correctly express CPS behavior and be able to perform model-based design, synthesis, and verification.

Object-oriented (OO) design enables hierarchical design with method call interfaces that provide re-usability, scalability and interface abstractions. OO design can be used in distributed applications, where the focus is on static structure, a breakdown of the architecture into a collection of *objects* with a procedure call interface. But the interaction between these objects is left to the method interactions and no clear computation model is explicitly defined. CPS design, in contrast, requires interaction models between components that are independent from internal component semantics. Actor-oriented programming builds upon the philosophy of OO design, but it extends the idea of functional abstractions further by defining *actors* and giving a clear semantics to their interactions with each other [11].

Many examples of actor-oriented tools exist today. Frameworks such as Simulink, LabVIEW and Ptolemy II [13] adopt actor oriented design methodologies and are widely used for design of embedded systems. Actor-oriented design in Ptolemy II has been studied in the context of several CPS domains including smart grid, aircraft, and automotive systems [3].

Tracking the developments in distributed computing and embedded systems, closed-loop CPS applications on sensor networks have become more feasible in the last decade. Various mobile sensor network applications have become widely deployed, triggering case studies in the areas of surveillance [5], hazard detection [15], and cooperative search [7].

3. APPLICATION REQUIREMENTS

A fundamental challenge in distributed CPS design is the gap between computation requirements and existing design tools for adaptive real-time simulation and deployment. Closed-loop data intensive robotic applications bring about unique requirements for such tools:

- Unlike off-line inference and control problems, robotic sensor network applications are subject to variable availability of sensor data and network resources due to the cyber-physical setting they operate in. Effects such as imperfect communication, anomalies, and latency, which may require dynamic adaptation, must be considered.
- Programming of robot swarms usually follows a workflow that relies of system design that considers the state-space model of the problem and the inference and optimization algorithms used for state estimation and control to be separate entities.
- In swarm applications, especially for a ubiquitous computing scenario, available computation resources may widely vary. This creates a demand for programming abstractions that can accommodate trade-offs between accuracy and timing, without requiring a reprogramming of the sensor network.

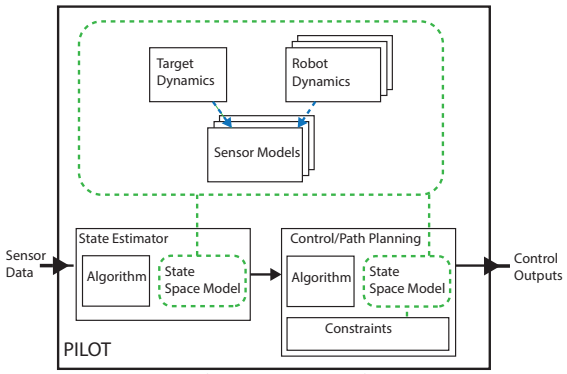


Figure 1: State-Space Aware System Architecture in PILOT

To address these requirements of streaming sensor network applications, we present PILOT, an actor-oriented machine learning and optimization toolkit, which is prototyped in the Ptolemy II framework. The following section will focus on presenting PILOT's system architecture, and an introduction to design of robotic sensor network applications using PILOT.

4. SYSTEM ARCHITECTURE

4.1 Actor-oriented Design

Ptolemy II is an open-source heterogeneous design framework that leverages actor-oriented design [9]. *Actors* in Ptolemy are concurrent components that communicate via messages that are sent and received through input and output ports. An actor interface is defined by a set of ports and parameters, and the semantics of communication between actors is mediated by a *director* that implements a specific model of computation. Ptolemy II enables hierarchical heterogeneous composition of computation models with a clearly defined compositional semantics. Some widely used models of computation in Ptolemy II include Process Networks (PN), Synchronous Data Flow (SDF) and Discrete-Event (DE). Also, a *decorator* actor design pattern has been implemented in Ptolemy. Similar to the concept in OO design, a `Decorator` actor is one that adds objects into a family of other actors.

4.2 Software Architecture

PILOT (Ptolemy Inference, Learning, and Optimization Toolkit) is developed based on the Ptolemy platform and consists of multiple Java packages that enable Bayesian inference, actor-oriented optimization and state-space modeling in an actor environment. Some of these aspects will be explained in detail in upcoming sections of the paper.

PILOT utilizes Ptolemy's `decorator` mechanism to implement state-space models and sensor models that *decorate* inference and optimization actors to give them operational meaning. Classes of decorator and analysis actors will be studied next.

4.3 Developing State-Space Aware Inference and Optimization Models

4.3.1 State-Space Models

We consider a finite state space system, where the system state is observable via a set of noisy observations, and the state dynamics satisfy the Markov property. A general representation of such model is given by

$$x_0 \sim \pi_X(x_0) \tag{1.1}$$

$$\mathbf{z_t}|x_t \sim g(x_t, u_t, t) \tag{1.2}$$

$$x_{t+1}|x_t \sim f(x_t, u_t, t) \tag{1.3}$$

where x_t and $\mathbf{z_t}$ correspond to the unknown state of a dynamic target and the set of observations at time t respectively, $\pi_X(\cdot)$ is the prior distribution of state x, $g(\cdot)$ denotes the stochastic measurement model as a function of state x_t, control inputs $u_t \in \mathcal{U}$, where $\mathcal{U}$ is the domain of control inputs. $f(\cdot)$ is the random function specifying the discrete state dynamics.

4.3.2 State-Space Modeling in PILOT

A state-space model implemented by the `StateSpaceModel` actor is depicted in Figure 2. This actor defines a target dynamics in 2-D space. A sensor model, `GaussianSensorModel`, is defined, which delivers a sensor measurement as a noisy estimate of the target position in 2-D space. The association of the target and the sensor is enabled by the decorator mechanism. It is seen in the same figure that the sensor is associated with the `StateSpaceModel` actor and is able to utilize the parameters provided by it (the state variables `x` and `y` in this case). Note that both of these actors are also decorators, which means, they are designed to provide parameters to other actors in the system design, i.e., state-space dependent state estimation and optimization actors. The use of this mechanism will be explained further in the following sections.

4.3.3 State Estimation

The general Bayesian state estimation problem aims at estimating the marginal posterior distribution $p(x_t|\mathbf{z}_{0:t})$ at time t, which is known as the *filtering* problem. Here, $p(x_t|\mathbf{z}_{0:t})$ is the PDF of x at time t given measurements $\mathbf{z}$, from time 0 to t. For a subset of the SSMs, in which $f(\cdot)$ and $g(\cdot)$ are linear functions with Gaussian noise distributions, the estimation problem can be delegated to a Kalman filter, which yields the optimal solution in the minimum mean-square error sense [4]. Variants of the Kalman filter for nonlinear state transition and measurement models exist, e.g., the Extended Kalman Filter (EKF). A more general Bayesian filtering approach, known as particle filtering, fits to a wide set of noise and measurement models.

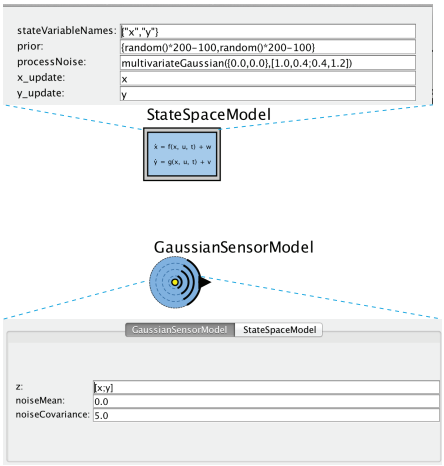


Figure 2: Sensor and Dynamics Modeling in PILOT

Particle filtering is a sequential Monte Carlo method that approximates a posterior distribution $p(x_t|\mathbf{z}_{1:t})$ with a weighted particle set, where each *particle* is a candidate state estimate, its weight being proportional to the likelihood. The filter approximates the posterior state with a probability mass function estimate, $\hat{p}(\cdot)$, given by

$$\hat{p}(x_t) = p(x_t|\mathbf{z}_{1:t}) = \sum_{i=1}^{N} w_t^i \delta(x_t - \tilde{x}_t^i)\,, \tag{2}$$

where $\tilde{x}_t^i$ denotes the i'th particle for the state estimate at time t, $\delta(\cdot)$ is the Dirac delta function and w_t^i is the weight associated with the particle i such that $\sum_{i=1}^{N} w_t^i = 1$, where N is the total number of particles. This approximate probability mass yields the minimum mean square error (MMSE) estimate of x_t as $\hat{x}_t^{MMSE} = \mathbb{E}(x_t) \approx \sum_{i=1}^{N} w_t^i \tilde{x}_t^i$.

It is important to highlight that the choice of a particle filter over parametric estimation methods like Kalman filters in actor-oriented settings is that downstream actors have direct access to particles, and would have direct control over tuning the subset of particles used in approximate algorithms. The ability to do so is important in a ubiquitous computing scenario, for which varying computational resources may require on-line algorithms to adapt their resources to compromise accuracy for better real-time performance.

The factoring of sensor models, target dynamics and the algorithms being used is a key feature of PILOT. As demonstrated in Figure 1, the software architecture enables state estimation and optimization actors to be *decorated* by state-space models defining target dynamics and sensor measurements.

State estimation actors in PILOT are designed to completely be defined by the decorators they are associated with. Associating a state estimator with a given state space model will completely define the state estimation problem. The specific *algorithm* used, e.g., particle filtering, Kalman filtering, will be orthogonal to the system design.

4.3.4 Constrained Optimization for On-line Control Applications

Next, we present an actor-oriented approach to performing *optimization* on streaming data. Consider the general optimization problem

$$\begin{aligned} &\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}}\ f(\mathbf{x}, \mathcal{Q}) \\ &\text{subject to } \mathbf{g}(\mathbf{x}, \mathcal{Q}) \geq 0\,, \end{aligned} \tag{3}$$

where $f(\cdot)$ is the objective function, $g(\cdot)$ is a vector-valued constraint function, and $\mathcal{Q}$ is a vector of function parameters.

As part of PILOT, we introduce an actor interface called `CompositeOptimizer` which enables $f(\cdot)$ and $g(\cdot)$ to be defined as *actors* that operate on input tokens, and upon firing, produce a scalar value for the objective function and a constraint vector evaluated at $\mathbf{x}$.

The operational semantics of the actor is given by Algorithm 1. The `CompositeOptimizer` is an actor whose internal execution semantics is governed by the `OptimizerDirector`. The current implementation of the director includes

Algorithm 1 `CompositeOptimizer`

Input: $\mathcal{Q} \Leftarrow \mathcal{Q}_i$
Output: $\mathbf{x}^*$ that is a local optimum of $f(\cdot)$
define P: An actor that implements SDF semantics and has inputs: $\mathbf{x}, \mathcal{Q}$ and outputs: f, g
while $k < k_{MAX}$ & !CompositeOptimizer.converged() **do**
$\mathbf{x}^{(k)} \Leftarrow$ OptimizerDirector.getNextX();
P.readInputs($\mathbf{x} \Leftarrow \mathbf{x}^{(k)}$, $\mathcal{Q} \Leftarrow \mathcal{Q}_i$);
P.execute();
P.writeOutput($f(x^{(k)}, \mathcal{Q}_i) \Rightarrow f^{(k)}$, $g(x^{(k)}, \mathcal{Q}_i) \Rightarrow g^{(k)}$);
OptimizerDirector.computeNextX($f^{(k)}$,$g^{(k)}$);
end while
$\mathbf{x}^* \Leftarrow$ CompositeOptimizer.getOptimalX();

a direct method solver, namely, Constrained Optimization By Linear Approximation (COBYLA) [12], as well as a barrier interior-point method based solver [2]. The methods `getNextX()` and `getOptimalX()` in Algorithm 1 are implemented by the specified solvers.

The `CompositeOptimizer` instance, as presented in Figure 3, is a sample realization of the actor that performs the streaming optimization problem given by

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}, q_1, q_2)$$
$$\text{subject to } g(\mathbf{x}, q_1, q_2) \geq 0\,, \tag{4}$$

where $f(\cdot)$ and $g(\cdot)$ are evaluated by executing actors `f(x)` and `g(x)` with inputs $\{\mathbf{x}^{(k)}, q_1(i), q_2(i)\}$ at each iteration, where the value of optimization variable $\mathbf{x}$ at each gradient descent step iteration is indexed by k, and is determined internally by the solver.

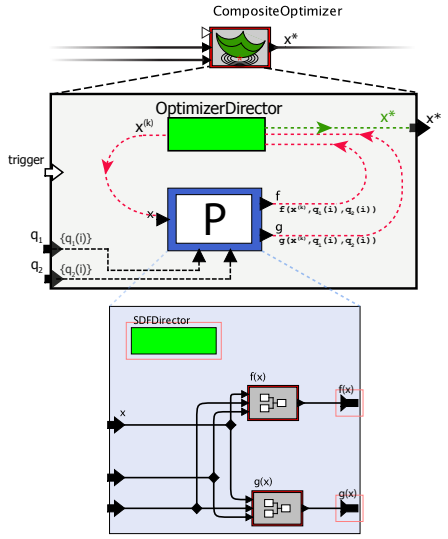


Figure 3: Operational Semantics of `CompositeOptimizer`

5. CASE STUDY: COOPERATIVE MOBILE SENSOR NETWORK CONTROL

5.1 Problem Statement

Emerging application scenarios on robotic sensor networks include cooperative target localization and tracking, Simultaneous Localization and Mapping (SLAM), and obstacle avoidance. The use of information theoretic objectives for sensor network management has been a recent area of interest. The reader is referred to [16] for a complete overview of information theoretic metrics for control purposes.

In this case study, we will consider a cooperative multi-robot target localization scenario using robots equipped with range-only sensors. A particle filtering algorithm is used to perform target state estimation. The goal of the mobile sensor network is to successfully locate and track target position, and future goals will include pursuit objectives.

The target search problem can be structured with the following state-space model

$$x_0 \sim \pi_X(x_0) \tag{5.1}$$
$$z_{j_t} = \|R_{j_t} - x_t\| + \nu_t \tag{5.2}$$
$$x_{t+1} = x_t + \omega_t\,, \tag{5.3}$$

where (5.2) corresponds to a range measurement made by robot j, z_{j_t} denotes the observation made by the j'th sensor at time t, R_{j_t} is the true position of robot j at time t and ν_t is a random variable defining measurement noise.

The unknown state x is the target position in $\mathbb{R}^2$, for our case study. The state dynamics of the uncooperative mobile target is assumed to be unknown to the application, and is characterized by a random walk with process noise defined by ω_t. State estimation of the unknown target will be performed given measurements from a set of robots, and will be represented as a set of particles, $\{w_t^i, \tilde{x}_t^i\}_{i=1}^N$.

5.2 State-Space Aware Application Design

The traditional approach to implementing an end-to-end system for designing an end-to-end target tracking application defined by (5.1)-(5.3) follows the monolithic approach of implementing a state estimator, followed by a controller, which are both implicitly dependent on the state-space of the problem.

PILOT's approach to designing such application differs from the traditional approach as it exploits the dependence of algorithmic blocks on the underlying problem state-space by defining *explicit* interfaces to the system dynamics.

The end-to-end application demonstrated in Figure 6 enables the user to define `RobotDynamics`, `TargetDynamics` and sensor models as actor interfaces. The target dynamics as given by (5.1) and (5.3) are contained by the `TargetDynamics` actor. The range sensors defined by (5.2)) are implemented by the `RangeSensor` actor, which is decorated by the `TargetDynamics` state-space model and therefore, can utilize the state-space parameters defined by the target and provide an imperfect measurement based on its parameters.

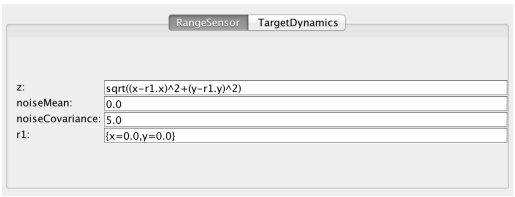
The usefulness of this interface definition is not limited to defining system dynamics. The architecture also enables learning and optimization actors to only include algorithm-specific implementations, and not problem-specific details. For instance, the `ParticleFilter` and `MutualInformation-Approximation` actors that are part of the estimation and control model given in Figure 7 are decorated by the `Range-Sensor`, and `TargetDynamics` decorators. The decorator association gives the actors operational meaning and defines the state-space aware problem. The user interface for the decorator association process is shown in Figure 4.

5.3 Simulation Setup

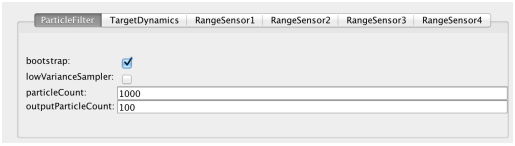
The case study assumes a network of robots equipped with range-only sensors. Simulation parameters are summarized in Table 1.

Simulation Parameter	Value
Size of Robot Team (M)	4
Search space	200x200 units
Sensor measurement noise	$\nu_t \sim \mathcal{N}(0, 5.0)$
Maximum Robot Speed	20 units/s
Iteration Frequency	10 Hz
Target Dynamics	Circular Motion with $\omega = \pi/5$
Target Position Prior (π_X)	Uniform over search space
Initial Target Position	$\mathbf{I}\|_{t=0} = \begin{bmatrix} -50 & 50 \end{bmatrix}$
Robot Control Inputs	$u_t^{(i)} = \begin{bmatrix} v_x & v_y \end{bmatrix}$, $i \in \{0, 1, 2, 3\}$
Robot Dynamics	$R_{i\,t+1} = R_{i\,t} + u_t^{(i)} \Delta t$, $i \in \{0, 1, 2, 3\}$

Table 1: Simulation Parameters



(a) `RangeSensor` Actor Parameters



(b) `ParticleFilter` Actor Parameters

Figure 4: PILOT State-Space Model User Interfaces

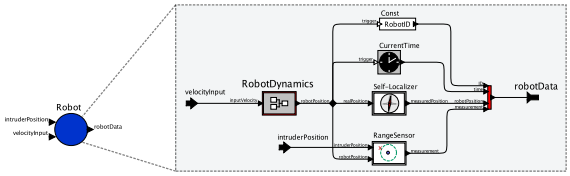


Figure 5: Model of the Robot Equipped with Range Sensor

The robot model is given in Figure 5. Note that in this model, `Robot` actors include models of range sensors with additive noise, making measurements to a target and delivering this data to a centralized computation unit. The refinement of the Target State Estimation and Control actor is depicted in Figure 7, which is an end-to-end PILOT model that includes state-space models of robot sensors, target dynamics, state estimation and trajectory optimization.

5.4 Actor-Oriented Implementation of Control Policies

Mutual information between target state and robot measurements can be used to derive a control rule to maximize the expected future mutual information, i.e., to minimize the expected future uncertainty in the target location estimate. The observers in this case are mobile robots with navigation, that expect velocity inputs from a centralized controller. The proposed objective function is given by

$$\mathbf{u}^* = \arg\max_{\mathbf{u}_\tau} I(z_\tau; x_\tau) \tag{6.1}$$

$$s.t.\ ||u_t^{(i)}|| \leq V_{max},\ i = 1, 2, ..., M \tag{6.2}$$

$$I(\mathbf{z}_\tau; x_\tau) := H(\mathbf{z}_\tau) - H(\mathbf{z}_\tau | x_\tau)\,, \tag{6.3}$$

where $I(\cdot;\cdot)$ is the mutual information (MI) defined between two random variables that are its arguments, $H(\mathbf{z}_\tau)$ is the entropy of the measurement set $\mathbf{z}_\tau$, $H(\mathbf{z}_\tau|x_\tau)$ is the conditional entropy of the measurements given the state belief, $\tau = [t+1, ..., t+T]$, where T is the time horizon of the control problem, and $\mathbf{u}$ is the array of control inputs to the mobile sensors.

We now illustrate how PILOT enables efficient exploration of a variety of control policies. Figure 7 describes a `CompositeOptimizer` interface that is configured to compute MI, given a set of particles and robot positions. The actor named `MutualInformationApproximation`, decorated by the `RobotDynamics` state-space definition that explains the dynamics of search robots, computes an approximate MI between a particle set and robot trajectories.

A subset of common control policies for robotic path planning, as given by Table 2, have been implemented using PILOT. Execution traces for each implemented trajectory control policy is given in Figure 8, which assume the parameter space given by Table 1. Note that the path planning problem defined by the PILOT model only depends on the choice of the objective function and constraints, and requires no modification to the state-space model itself.

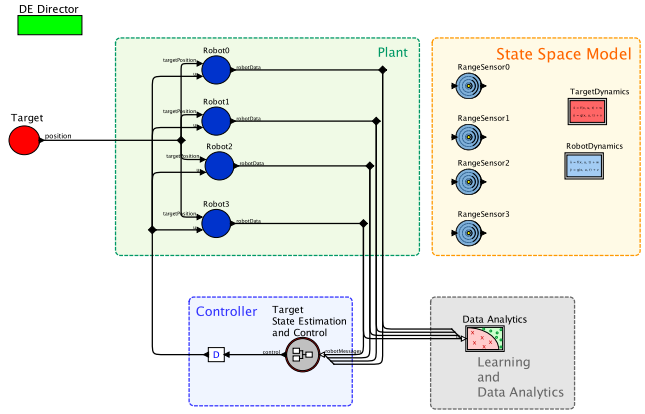


Figure 6: Top-Level Model for Range-Only Target Localization

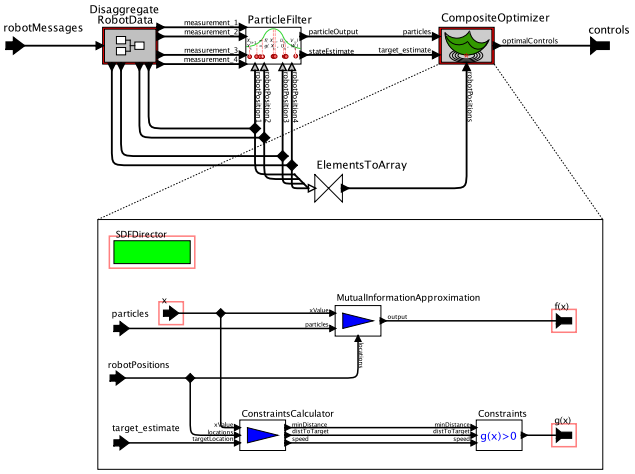


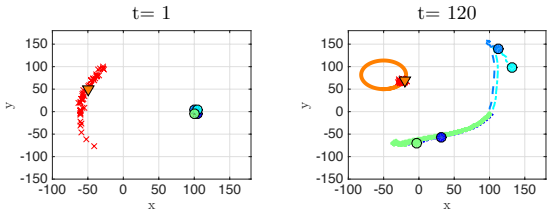
Figure 7: PILOT Model for Target State Estimation and Trajectory Optimization

6. CONCLUSION

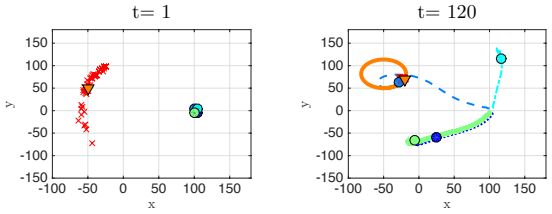
We presented PILOT, which is a novel toolkit for actor-oriented design of machine learning and optimization algo-

Control Policy	Optimization Problem
MI Maximization	$\mathbf{u_t}^* = \arg\max_{\mathbf{u}_t \in \mathcal{U}^M} I(\mathbf{z}_{t+1}; x_{t+1})$ s.t (6.2) holds
MI Maximization with Single Pursuer	$\mathbf{u}_t{}^{(i)*} = \begin{cases} \arg\min_{\mathbf{u}_t{}^{(i)} \in \mathcal{U}} \|\mathbf{R}_{t+1} - x_{t+1}\| \\ \text{s.t (6.2) holds} \quad \text{if } d_t{}^{(i)} < d_t{}^{(j)}, \forall j \neq i \\ \arg\max_{\mathbf{u}_t{}^{(i)} \in \mathcal{U}} I(z_{t+1}^{(i)}; x_{t+1}) \\ \text{s.t (6.2) holds} \quad \text{otherwise} \end{cases}$ $d_t{}^{(i)} := \|R_{i\,t} - x_t\|,\ i \in \{1, 2, ..., M\}$
Direct Target Pursuit	$\mathbf{u_t}^* = \arg\min_{\mathbf{u}_t \in \mathcal{U}^M} \|\mathbf{R}_{t+1} - x_{t+1}\|$ s.t (6.2) holds

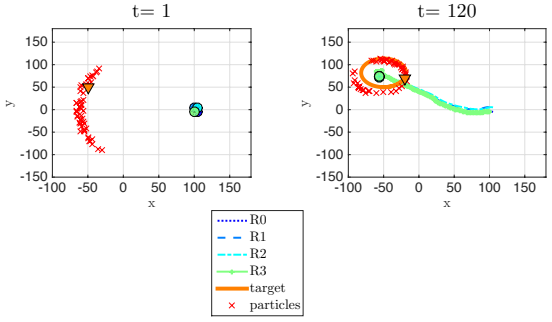
Table 2: Control Policies for Path Planning



(a) Sample trajectory for the MI Maximization Control Policy



(b) Sample trajectory for MI Maximization with Single Pursuer



(c) Sample trajectory for Direct Target Pursuit

Figure 8: Simulation Traces for Control Policies given by Table 2

rithms on streaming data for robotic sensor network applications. We demonstrated an actor interface to a collection of algorithms, and described how estimation and control algorithms can be customized based on state-space models of the problem space. We illustrated the use of the toolkit on a case study of cooperative robotic sensor network target localization with state-space dependent on-line target state estimation with a variable set of sensor inputs. Future work includes extending the framework to capture additional machine learning techniques to perform on-line classification for anomaly detection and model predictive control, as well as exploring model-predictive control capabilities using PILOT.

7. REFERENCES

[1] N. Bezzo, J. Park, A. King, P. Gebhard, R. Ivanov, and I. Lee. Poster abstract:ROSLab-A modular programming environment for robotic applications. In *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*, pages 214–214. IEEE, 2014.

[2] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[3] P. Derler, E. A. Lee, and A. S. Vincentelli. Modeling cyber–physical systems. *Proceedings of the IEEE*, 100(1):13–28, 2012.

[4] A. Doucet and A. M. Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 12:656–704, 2009.

[5] B. Grocholsky, J. Keller, V. Kumar, and G. Pappas. Cooperative air and ground surveillance. *Robotics & Automation Magazine, IEEE*, 13(3):16–25, 2006.

[6] B. Hartmann, S. R. Klemmer, M. Bernstein, L. Abdulla, B. Burr, A. Robinson-Mosher, and J. Gee. Reflective physical prototyping through integrated design, test, and analysis. In *Proc. of User interface software and technology*, pages 299–308. ACM, 2006.

[7] G. M. Hoffmann and C. J. Tomlin. Mobile sensor network control using mutual information methods and particle filters. *Automatic Control, IEEE Transactions on*, 55(1):32–47, 2010.

[8] E. A. Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363–369. IEEE, 2008.

[9] E. A. Lee, S. Neuendorffer, and M. J. Wirthlin. Actor-oriented design of embedded hardware and software systems. *Journal of circuits, systems, and computers*, 12(03):231–260, 2003.

[10] Y. Li, H. Lu, and H. Zhang. Optimistic programming of touch interaction. *ACM Trans. Comput.-Hum. Interact.*, 21(4):24:1–24:24, Aug. 2014.

[11] J. Liu, J. Eker, J. W. Janneck, X. Liu, and E. A. Lee. Actor-oriented control system design: A responsible framework perspective. *Control Systems Technology, IEEE Transactions on*, 12(2):250–262, 2004.

[12] M. J. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in opt. and num. analysis*, pages 51–67. Springer, 1994.

[13] C. Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

[14] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[15] M. Schwager, P. Dames, D. Rus, and V. Kumar. A multi-robot control policy for information gathering in the presence of unknown hazards. In *Proc. of the Int. Symp. on Robotics Research (ISRR 11)*, 2011.

[16] J. L. Williams. *Information theoretic sensor management*. PhD thesis, Massachusetts Institute of Technology, 2007.