AFRL-AFOSR-VA-TR-2020-0026

Enabling Robust Persistent Autonomy in Robots

Ross Knepper
CORNELL UNIVERSITY

05/22/2020
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTA2
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
22-05-2020	Final Performance	01 Jan 2017 to 31 Dec 2019

4. TITLE AND SUBTITLE
Enabling Robust Persistent Autonomy in Robots

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-17-1-0109

5c. PROGRAM ELEMENT NUMBER
61102F

6. AUTHOR(S)
Ross Knepper

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
CORNELL UNIVERSITY
373 PINE TREE RD
ITHACA, NY 14850-2820 US

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AF Office of Scientific Research
875 N. Randolph St. Room 3112
Arlington, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/AFOSR RTA2

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-AFOSR-VA-TR-2020-0026

12. DISTRIBUTION/AVAILABILITY STATEMENT
A DISTRIBUTION UNLIMITED: PB Public Release

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The long-term objectives of this project are to increase the level of autonomy of robots by giving them an introspective capability. Introspection involves modeling the robot's own behavior and using that model to reflect on how to change its behavior in response to unanticipated events. As a part of this project, we have developed predictive models of a number of robot models for testing these algorithms, including an autonomous car, an autonomous airship, a pedestrian robot navigating among people, and a manipulation scenario involving cloth folding. We made use of these models to demonstrate three primary goals: failure handling, mission planning, and adaptable specification. Recommendations for future research directions in persistent autonomy are included.

15. SUBJECT TERMS
Persistent Autonomy, Artificial Intelligence

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			LAWTON, JAMES
Unclassified	Unclassified	Unclassified	UU		19b. TELEPHONE NUMBER *(Include area code)* 703-696-5999

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
30-03-2020	Final Report	From 1-1-2017 - To 31-12-2019

4. TITLE AND SUBTITLE

Enabling Robust Persistent Autonomy in Robots

5a. CONTRACT NUMBER

FA9550-17-1-0109

5b. GRANT NUMBER

FA9550-17-1-0109

5c. PROGRAM ELEMENT NUMBER

N/A

6. AUTHOR(S)

Ross A. Knepper

5d. PROJECT NUMBER

N/A

5e. TASK NUMBER

N/A

5f. WORK UNIT NUMBER

N/A

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Cornell University
OFFICE OF SPONSORED PROGRAMS
373 PINE TREE RD
ITHACA NY 14850-2820

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AF OFFICE OF SCIENTIFIC RESEARCH
875 NORTH RANDOLPH STREET, RM 3112
ARLINGTON VA 22203-1954

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The long-term objectives of this project are to increase the level of autonomy of robots by giving them an introspective capability. Introspection involves modeling the robot's own behavior and using that model to reflect on how to change its behavior in response to unanticipated events. As a part of this project, we have developed predictive models of a number of robot models for testing these algorithms, including an autonomous car, an autonomous airship, a pedestrian robot navigating among people, and a manipulation scenario involving cloth folding. We made use of these models to demonstrate three primary goals: failure handling, mission planning, and adaptable specification. Recommendations for future research directions in persistent autonomy are included.

15. SUBJECT TERMS

robotics, autonomy, unmanned vehicles, planning, control, sensing, persistent monitoring, persistent autonomy, failure handling

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Ross Knepper
U	U	U	UU	34	19b. TELEPHONE NUMBER *(Include area code)* (607) 255-5014

Enabling Robust Persistent Autonomy in Robots

Ross A. Knepper, PI
Cornell University
Gates Hall
Ithaca, NY 14853
Phone: (607) 255-8634
email: rak@cs.cornell.edu

Award Number: FA9550-17-1-0109

1 Summary

The long-term objectives of this project involve increasing the level of autonomy of robots by giving them an introspective capability. Introspection involves modeling the robot's own behavior and using that model to reflect on how to change its behavior in response to unanticipated events. As a part of this project, we have developed predictive models of a number of robot models for testing these algorithms, including an autonomous car, an autonomous airship, a pedestrian robot navigating among people, and a manipulation scenario involving cloth folding. We made use of these models to demonstrate three primary goals.

First, we seek to develop intelligent failure handling techniques that can diagnose and correct diverse types of failures using introspection. Second, we want to perform intelligent, long-term mission planning that plans ahead to make best use of data collected during future mapping and surveillance activities. Third, we aspire to achieve intelligent execution adaptation to prior specifications given novel environments encountered by a quadcopter. Four conference papers and two workshop papers have so far been published from the work on this grant, with one final conference paper currently under review and another in preparation. Thirteen researchers were funded to work on this project.

This project represents a very preliminary investigation into the problem of persistent autonomy. We therefore offer recommendations in Section 5 for future directions to pursue for persistent autonomy research, including an

extensive debrief of broader lessons learned on the current project.

2 Introduction

As practiced today, robotics is the process of making increasingly greater simplifying assumptions about the world until we are able to specify and solve a problem in hardware and software. We note how often research robots appear brittle in their behavior and observe that this results in part from inaccurate assumptions baked into engineered solutions.

The persistent autonomy problem asks us to build robots and AI systems that continue to function robustly for prolonged periods of time without human intervention to correct faults. Persistent autonomy requires higher-level problem-solving and decision-making skills than robots and AI today possess. In fact, the needs of the persistent autonomy problem remain poorly mapped out. This report seeks to examine more broadly some of the technological solutions that are needed before we will see a robust persistent autonomy take shape in the world.

Very little work has been done on the problem of persistent autonomy. The biggest achievement to date in experimental persistent autonomy is the three-year EU project PANDORA, which culminated in a demonstration of an autonomous underwater vehicle manipulating valves for three hours without human help [10]. The current project seeks to build technologies that enable longer-term autonomy over more diverse tasks compared to PANDORA. It does not attempt a prolonged empirical demonstration but rather examines the fundamentals of persistent autonomy.

Project Goals. In this project, we pursue three goals pertinent to autonomous vehicle navigation.

First, we examine the use of introspective models in failure handling. The challenge is that most failure handling in robotics requires an exhaustive set of fairly hand-coded routines to handle each modality of failure. In the navigation problem, we have developed a capability to detect failures that does not require knowledge of all possible failure modalities. We have also showed that a robot can react in response to its prediction to generate desirable behavior.

Second, we examine the problem of long-term mission planning for mobile surveillance robots like the autonomous airship. Given a partial map of some interesting property like vegetation health, we formulate information

gain during exploration as a submodular function and show that bounded-suboptimal exploration is efficiently achievable.We then use matrix completion techniques to fill in the gaps not explored so that the robot can make intelligent decisions about where to go next to collect data. Maps might for example be sourced from satellite data with occlusions due to cloud cover or be limited to observations of traces of the robot's path. After filling in the blanks in a manner that maximizes extrapolation from the available data, we plan to use the estimates to pick routes for future data gathering in order to supplement the available information and maximize information gain in the map.

Third, we investigate the problem of writing flexible behavior specifications that adapt to uncertain future environments. Taking inspiration from human's ability to perform abstract specifications, we examined a natural language specification interface. We showed a quadcopter responding with flexibility to novel environments and unfamiliar landmarks while attaining navigational objectives on the fly. We consider the strengths and weaknesses of a statistical learning approach to this problem.

Technical Objectives. Work in this project proceeding along three parallel tracks, united by the use of predictive vehicle models.

In failure handling, the objective is to use techniques from model-predictive control (MPC) to introspect on the behavior of a vehicle motion model for detection, diagnosis, and remedy of failures. We focus here on failures that affect the (perceived) motion of the vehicle, including motor speed and position, sensor calibration, and slip. By comparing its trajectory to previous predicted trajectories, the robot can measure the magnitude of deviations. Deviations from the predicted trajectory arise due to unmodeled disturbances, including failures. By simulating hypothetical state changes, the robot can then explore counterfactuals and by so doing attempt to diagnose and repair the failure.

In mission planning, we are building maps with holes in the data, such as cloud-cover in satellite data, normalized difference vegetation index (NDVI) data over time for estimating drought, or vegetation classification algorithms based on transects from aerial vehicle flyovers. We use low-rank matrix-completion techniques [15] to fill in the gaps. We will then plan routes through the map to maximize information gain according to the confidence with which various regions can be predicted.

In adaptable specification, we explore the merits of natural language as a specification tool. When exploring a novel environment, the robot must be

able to ground the specifications in the context of that environment in order to correctly interpret the specified behavior. We experiment with a statistical method designed to recapitulate the traditional robot planning pipeline. We consider the costs and benefits of both approaches and conclude that both are insufficient.

3 Methods, Assumptions, and Procedures

In this work we assume there exists a dynamics model of the form

$$\ddot{x} = f_y(x, u, t).$$

Suppose x_t designates some measured state at time t. Given some initial condition (x_t) and parametrized control input (u), we can integrate this model (f) through some short period of time (t) in order to predict future robot behavior ($\ddot{x}$). The model f is parametrized by a set of latent variables y that describe physical and environmental operating properties and state (for example, the coefficient of friction between the wheels and ground). Of course the prediction resulting from integrating this model is only as good as its inputs. Whether it is implemented from a statistical summarization of measured behavior or via a physics simulator built on top of Newton's laws of motion, we can expect that the prediction gets less reliable further into the future due to the accumulation of error. All of the work described here starts from this assumption and explores various approaches to improve the robot's performance without reliance on human help to tune or adjust the model or the physical system. Since we presume that the prediction has a balance of information and noise, the objective is to leverage the predictions without overcommitting to any one prediction during the course of execution. By making repeated predictions with varying initial conditions, control inputs, or latent variables, the robot can gain information to increase the robustness of its behaviors.

3.1 Failure Handling

We use prediction at a small time-scale to study responses to failure. We define failure as an unexpected and deleterious event that affects the robot in performance of its task. Failures may concern the current task or the robot itself.

The reality of operating a complex engineered system such as a robot is that the probability distribution of possible robot failures has a long tail of extremely rare events. The integral of these rare events constitutes a significant fraction of the overall probability mass. Thus, approaches that explicitly rely on past experience are inadequate.

Our approach employs the dynamics model to evaluate counterfactuals in several phases. We generate these counterfactuals by integrating the model from different start times, assuming different latent variables, and using different control inputs.

- **Phase 1: Detection.** Suppose that state was measured at times $t-\epsilon$ and t. Our model can be used to predict the latter time from the former using the model. Substantial deviation from nominal behavior suggests that the model is making bad predictions.

$$\left| x_t - \int_{t-\epsilon}^{t} \int_{t-\epsilon}^{t} f_y(x, u, t)\, \mathrm{d}t\, \mathrm{d}t + x_{t-\epsilon} \right| > v_{\text{thresh}},$$

  where v_{thresh} is a threshold value. This triggers a detection event.

- **Phase 2: Diagnosis.** Within the context of the dynamics model, there are a few causes that could trigger a detection. One possibility is that the state measurements $x_{t-\epsilon}$ and/or x_t are inaccurate due to sensing errors. Another possibility is that the latent variables y in the model are not accurately describing the current situation. This phase involves varying the control inputs u in a principled manner to gain information that helps narrow down the list of possible explanations. Our present aproach involves sampling from a low-dispersion sequence that covers the entire control space. The result of this phase is a list of experimental results enumerated from $i = 1 \ldots n$ and expressed as tuples $(t_s^i, x_s^i, u^i, t_f^i, x_f^i)$ – the start time and state, the control input, and the final time and state. Note that the latent variables and possible sensing errors that generated these results are unknown.

- **Phase 3: Explanation.** In this phase, we explore counterfactuals in an effort to discover what causes generate the observed behavior. This process involves a search over the hypothesis space of sensing errors and latent variables. Note that sensing errors can also be explained by a change in the latent variables. For simplicity here, we collapse

the hypothesis search space to only latent variables. Given the set of empirical results, we find the latent variables that minimize the sum of squared errors.

$$\underset{y'}{\operatorname{argmin}} \sum_{i=1}^{n} \left(\int_{t_s^i}^{t_f^i} \int_{t_s^i}^{t_f^i} f_{y'}(x, u^i, t)\, \mathrm{d}t\, \mathrm{d}t + x_s^i - x_f^i \right)^2$$

The best y' becomes our new hypothesis.

- **Phase 4: Repair.** Given the hypothesis y', we repair the failure by solving the optimization problem

$$\underset{u'}{\operatorname{argmin}} \left| \int_{t-\epsilon}^{t} \int_{t-\epsilon}^{t} f_y(x, u, t)\, \mathrm{d}t\, \mathrm{d}t - \int_{t-\epsilon}^{t} \int_{t-\epsilon}^{t} f_{y'}(x, u', t)\, \mathrm{d}t\, \mathrm{d}t \right|.$$

That is, if u was expected to produce optimal behavior given that y holds, then u' optimizes behavior under hypothesized conditions y'.

3.2 Long-Term Mission-Planning to Maximize Information Gain

Here we look at the long-term effects of predicting a dynamics model of information. We consider an aerial vehicle flying over a terrain performing some observation activity. Observations of two regions of the terrain are correllated according to some complex function. Examples include weather-related properties such as foliage and vegetation health, and human activity such as transportation via ground vehicles.

The problem is two-fold. First, given a set of past observations, develop a statistical model to predict the unseen regions of the map. Second, select a route through the terrain to maximize information gain and reduce uncertainty about future predictions. In essence then, we are modeling the information dynamics of exploration.

3.2.1 Problem Formulation

Let $\mathcal{R}$ be a grid-discretized two-dimensional region composed of L cells, or sampling locations. Let p be a continuous-valued environmental attribute that can be observed at every cell in $\mathcal{R}$. We assume the value of p at every cell in $\mathcal{R}$ is time-varying, but changes relatively slowly with respect to the

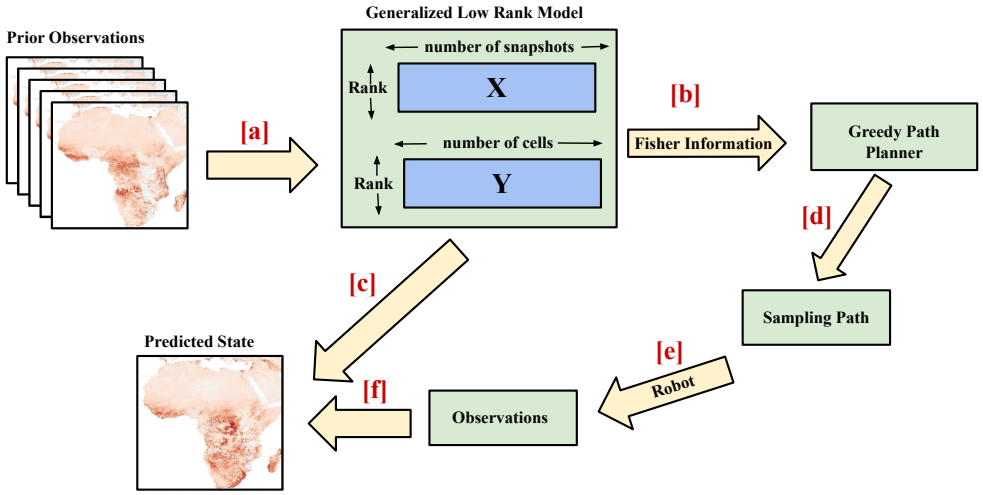

Figure 1: Prior observations are used to generate a generalized low rank model (arrow [**a**]). Fisher information computed from the GLRM (arrow [**b**]), is used by the path planner to form a sampling path (arrow [**d**]). The robot takes observations along the path (arrow [**e**]). The observations are used (arrow [**f**]), in conjunction with the GLRM (arrow [**c**]) to predict the states of the unobserved cells.

sampling frequency. For instance, the hue of foliage over the northeast region of the Americas in the fall can be considered static over the span of a few hours, but is dynamic over the span of a few days. Therefore, for this domain, observations taken within a few hours of each other are considered to be sampled at the same time.

We have a dataset containing a set of snapshots of p in $\mathcal{R}$ for T different times. Each snapshot contains observations of the value of p in a subset of $\mathcal{R}$. Let $C(P) : \mathcal{P}(\mathcal{R}) \rightarrow \mathbb{R}$, where $P \subseteq \mathcal{R}$, be a cost function from a subset of cells in $\mathcal{R}$ to a real value. This function computes the cost of traveling between a set of cells. Let $b \in \mathbb{R}$ be a budget corresponding with C.

We consider the problem of choosing a set of sampling locations, $P \subseteq \mathcal{R}$ such that: (1) $C(P) \leq b$, the cost of P is within the budget, and (2) the uncertainty about the state of p over $\mathcal{R}$ is minimized after observations are taken for all cells in P. We maximize Fisher information as a proxy for minimizing uncertainty. Additionally, we want to predict the current state of the unobserved cells in $\mathcal{R}$, given, ω, a recently-sampled set of observations of p in $\mathcal{R}$.

3.2.2 Approach

The prior observations are represented as a data matrix and used to fit a generalized low rank model (GLRM) [15] composed of matrices X and Y, as shown in Figure 1, arrow **[a]**. The GLRM captures the structure of p both spatially and temporally and is used to compute Fisher information for sets of cells in $\mathcal{R}$.

The greedy path planner uses Fisher information (Figure 1, arrow **[b]**), a cost function, and a cost budget to choose a set of sampling points in $\mathcal{R}$, starting an initial cell (Figure 1, arrow **[d]**). The Fisher information of the sampling points is within $\frac{1}{2}(1 - e^{-1})$ of optimal, given the constraints, and that the cost of visiting each point, per the cost function, is within the provided budget. The robot makes observations along this path (Figure 1, arrow **[e]**) to get ω, a set of observations.

Finally, ω and the GLRM (Figure 1, arrows **[f]** and **[c]**) are used to predict the states of the unobserved cells in $\mathcal{R}$.

Generalized Low Rank Model We represent the state of the attribute p over $\mathcal{R}$ using a generalized low rank model (GLRM). A GLRM is a vector model that represents large datasets with missing and noisy data as a pair of comparatively small, low rank matrices [15]. The low rank nature of this model means that it can predict unobserved values given a small number of accurate observations. In addition, Fisher information, representing the information of sets of cells as represented by the GLRM, can be computed using a generalized low rank model [16].

Assume the environmental variable p can be observed at L locations within $\mathcal{R}$ and observations are taken at T different times. Prior observations of p over $\mathcal{R}$ are given as a dataset. We collect these prior observations in a data matrix, $D \in \mathbb{R}^{T \times L}$, where the value of $D_{i,j}$ is the value of p observed at cell $i \in \mathcal{R}$ at the j^{th} time. Unobserved values over the T times and L cells are set to arbitrary values in D; their values do not affect the results. A projection function P_Ω is used to ignore the entries of D corresponding to missing or unobserved data. Let $\Omega \subseteq \{1, \ldots, T\} \times \{1, \ldots, L\}$ be the set of (time, location) tuples representing entries in D that contain observations. Let P_Ω be the projection operator that ignores the unobserved values in D.

We compute X and Y by

$$\underset{X \in \mathbb{R}^{k \times T}, Y \in \mathbb{R}^{k \times L}}{\operatorname{argmin}} \|P_\Omega(X^\mathsf{T} Y - D)\|_F^2,$$

where $k \in \mathbb{Z}$ is the rank of the model. Intuitively, the columns of Y are basis vectors for a latent space that encodes the spatial structure of p; each column corresponds to one location in $\mathcal{R}$. Similarly, the columns of X are basis vectors for a latent space that encodes the temporal structure of p. The product $X^\mathsf{T}Y$ is a minimal-error representation of the original data matrix, including predictions for unobserved points.

Path Planner We use a greedy path planner to choose the set of sampling locations, given an information measure, and cost function with a corresponding cost budget. The specific planning algorithm is introduced by Zhang and Vorobeychik and maximizes a submodular quantity while respecting a cost budget [17]. This planner requires a submodular function $\mathcal{I} : \mathcal{P}(\mathcal{R}) \to \mathbb{R}$ to maximize and an α-submodular approximate cost function $C : \mathcal{P}(\mathcal{R}) \to \mathbb{R}$ with a corresponding budget value. Each of these functions take a set of cells and returns a real number. The specific cost and submodular functions chosen for the monitor application are discussed below.

The planner selects a set of cells $P \subseteq \mathcal{R}$ from a given initial cell, s. The set P is initialized to contain the initial position, s, and the $k-1$ closest cells to s, where k is the rank of the GLRM. The extra cells are added because Fisher information is a meaningful quantity only if the number cells is be greater than or equal to the GLRM's rank.

At each iteration of the algorithm, the cell that maximizes the ratio of change in information to the change in cost is added to the set of sampling locations. Specifically, for each cell $x \in \mathcal{R}$, a set $P' = P \cup \{x\}$ is formed. If $C(P')$ is less than or equal to the budget, we compute:

$$\Delta_x = \frac{\mathcal{I}(P') - \mathcal{I}(P)}{C(P') - C(P)}.$$

Otherwise, if $C(P')$ is greater than the budget, x is not a valid option and Δ_x is set to zero. After iterating through each cell in the environment, a cell with the maximum value of Δ_x is added to the path P. This process is repeated until the maximum value of Δ_x is zero, that is, there is no cell to add to P that would respect the cost budget.

The path planner returns P, the set of cells in $\mathcal{R}$ where the robot must take observations of p. The cells in P are treated as waypoints, that is, the actual trajectory of the robot should pass through each cell in P, but the robot's trajectory is left to a lower level controller.

We use a greedy planning approach because Fisher information is submodular. Greedy approaches are computationally fast, simple to implement, and are known to approximately maximize submodular quantities. Zhang and Vorobeychik [17] prove that the value of the submodular function of the cells selected is within $\frac{1}{2}(1 - e^{-1})$ times of the maximum value of the submodular function possible subject to the the cost constraints.

Cost Function The cost function represents the limit on how far the robot can travel. We use a shortest path cost function and the nearest neighbor algorithm as the α-submodular approximate cost function, required by the planner. The nearest neighbor algorithm is a greedy-approximation for shortest path problems [5].

Starting with a given point, the algorithm adds points to the path based on proximity to the current point. That is, at each time-step, the next point to be traveled to is the one that is closest to the current point. Given a set of points $P = \{v_1, \ldots, v_n\}$, the nearest neighbor algorithm returns a travel ordering of these points, starting at v_1, the ordering is notated $P' = \{v'_1, \ldots, v'_n\}$. The cost of the path is:

$$C(V') = \sum_{i=1}^{n-1} d(v'_i, v'_{i+1}),$$

where $d(a, b)$ is the Euclidean distance between the points a and b.

For the planner's optimality guarantee to hold, the approximate cost function must be a $\psi(n)$-approximation, where n is the number of points in the set and $\psi(n)$ is some constant real number. A cost function is a $\psi(n)$-approximation when it is at most $\psi(n)$ times greater than the true cost. In addition, the cost function should be α-submodular. That is, when x, A, B are minimized and $A \subset B$ the quotient

$$\frac{cost(A \cup x) - cost(A)}{cost(B \cup x) - cost(B)}$$

is equal to α [17]. The nearest neighbor cost function is a $\psi(x)$-approximation and α-submodular [5].

Submodular Function Fisher information is used to quantify information for the planning algorithm. Fisher information is a measure of relative

information in a set of variables that can be computed directly from models, including the GLRM [4, 16]. By observing the value of p at the set of cell in $\mathcal{R}$ that maximizes Fisher information, the robot will maximize information gain in expectation.

Formally, the Fisher information of a set of points, V, is:

$$\mathcal{I}(V) = \log\det\left(\sum_{p\in P} Y_p Y_p^\mathsf{T}\right),$$

where Y is the matrix from the generalized low rank model and Y_p is the column of Y corresponding to the cell $p \in \mathcal{R}$. For the Fisher information to be a meaningful quantity, the number of points in the set must be greater than or equal to the number of rows in Y. Therefore we initialize the set of sampling points to include the $k-1$ nearest cells to the initial cell, where k is the rank of the GLRM. Fisher information is submodular because $\sum_{p\in P} Y_p Y_p^\mathsf{T}$ is a positive definite matrix and therefore Fisher information compatible with the greedy planning algorithm [6].

Region Completion After the robot collects observations, the GLRM is used to predict the unobserved cells of the region. Region completion allows the robot to predict the full state of the region, without fully observing it.

Let D, X, Y be as described in Section 3.2.2, then the values of the unobserved cells are a linear combination of the columns of Y, let x be the latent factor needed to determine the linear weights.

We view the set of observations made as a new row $d \in \mathbb{R}^L$ of D and a new column $x \in \mathbb{R}^k$ of X. Since the set of new observations do not cover the region, d has missing entries. We define the subset of $\mathcal{R}$ that was observed as $S \subseteq \{1, \ldots, L\}$. Define $Y_S \in \mathbb{R}^{k\times|S|}$ to collect those columns of Y corresponding to indices in S. This allows the unobserved cells to be ignored in the reconstruction process. In order to estimate x, we minimize x in the following:

$$\|P_S(Y^\mathsf{T}x - d)\|^2 = \|Y_S^\mathsf{T}x - d_S\|^2.$$

This can be reduced to:

$$(Y_S^\mathsf{T}Y_S)^\dagger Y_S^\mathsf{T} d_S,$$

where $A^\dagger$ represents the pseudoinverse of A. Note that the unobserved values are not included in the objective and therefore do not affect the solution.

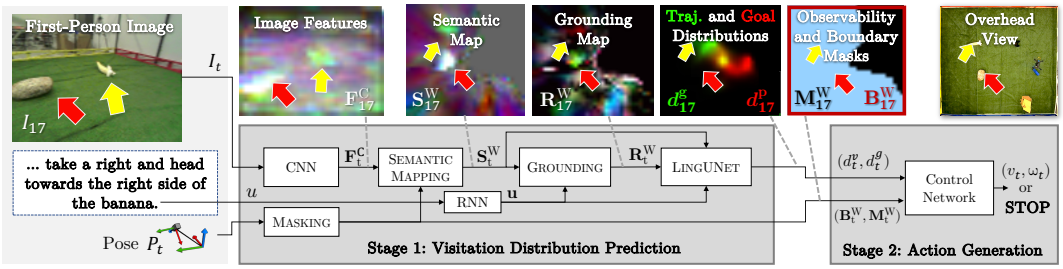

Figure 2: Model architecture illustration. The inputs are a camera view, natural language navigation specification, and pose (localization). Stage 1 discovers desirable locations in the map to visit (visitation distribution prediction), and Stage 2 utilizes the result of Stage 1 to compute actions for the quadcopter to execute in order to achieve the desired result. Note that only "Image Features" is computed in the camera frame; the remaining stages are in the global map frame. The system tracks which parts of the map it has seen so that it can reason about the possibility that referenced objects have not yet been seen. The red and yellow arrows indicate the rock and banana locations throughout the pipeline. An animated version of this figure is available at https://youtu.be/UuZtSl6ckTk.

Once x is computed, the values of p for the unobserved cells are predicted by $Y^\mathsf{T}x$.

3.3 Adaptible Specification

One attribute of natural language that makes it both difficult to use and powerful is the varying levels of detail with which specifications may be given.Although natural language may not seem like an obvious candidate for writing specifications for persistent autonomy, it provides one vital attribute: avoidance of overspecification.

Robots gain information from three sources: sensors, specifications, and implementation. Quite often, information about the robot's environment that in the most general implementation should originate from sensors originates instead from the implementation. In other words, the solution hard-codes too many assumptions. To study adaptabiity, we should therefore make the specifications and/or the sensor inputs more general. In this work, we do both by using natural language to instruct a quadcopter to navigate through novel environments via landmarks.

We use statistical learning to model this process not because it is any more general than traditional engineering approaches[1], but rather as a pro-

[1]Compared to engineered approaches, learning approaches trade off predictability regarding inputs unseen in training for generality across the breadth of possible inputs. Thus, they are not reliable as fielded systems but can be useful tools for exploring architectures.

totyping tool to investigate an architecture that could later be implemented in a traditional engineered approach. Such learning models can be quicker to build and explore before we commit to a set of engineered software modules. As such, this architecture (see Figure 2) is a proof of concept.

This implementation was built on a quadcopter with an architecture consisting of logical modules that have transparent meanings and could be constructed as engineered modules in order to avoid the drawbacks of statistical learning. The key architectural features three inputs (camera video, a natural language specification, and pose/localization) and two outputs (commanded linear and angular velocity of the quadcopter). Internally, it is organized into two stages. Stage 1 generates an approximate path that appears desireable based on the inputs over time. Stage 2 generates outputs in the form of actions for the quadcopter to execute, which optimize for both following the path and exploring unseen terrain.

The architectural modules of Stage 1 shown in Figure 2 are as follows. The first module computes Image Features in a first-person view based on familiarity with previously seen objects. Next, the current image features are projected onto a global map using the pose and a pinhole camera model. The results of that projection over time are integrated to generate a Semantic Map, which tracks locations of potential interest within the terrain. The integral causes it to remember locations that have gone out of view. Next, a natural language processing module filters the semantic map to produce a Grounding Map, in which only locations mentioned in the specification are included. At the end of Stage 1, a Trajectory Distribution and Goal Distribution are output. Combined, these predict a path which, when followed, produces behavior similar to the specification. The approximate nature of this path (a distribution) is a feature because it prevents the vehicle from over-constraining the path plan too soon. That is, the path plan gets continuously refined as the vehicle navigates.

Stage 2 includes two modules as shown in Figure 2. The first module tracks Observability and Boundary Masks. The second generates control signals that weigh exploration more highly when the path plan and goal are less certain because this indicates that the correct landmarks may not have been observed yet. When those distributions are more certain, exploitation is prioritized.

4 Results and Discussion

This project has so far generated seven publications [1–3, 8, 9, 11, 12]. During the period of performance, this award supported one faculty member (Ross Knepper: 3.385 person-months), three Ph.D. students (Valts Blukis: 5.625 person-months, Claire Liang: 4.5 person-months, Elizabeth Ricci: 4.5 person-months), and nine undergraduate students.

4.1 Failure Handling

We studied failure handling in several domains. One was the problem of mobile robot navigation among people in social environments. In this work, we implemented prediction and avoidance behaviors to prevent collisions and other socially incompetent outcomes. We used a dynamics model based on the Social Force model [7]. We showed that we can make accurate predictions and use those predictions to read humans' intent. We also showed that the robot can react appropriately to those intentions. We presented this work at RSS 2017 [8] and IROS 2017 [9].

A Masters of Engineering student studied failure handling both in simulation and on a real robot platform. First, she developed a high-fidelity physical car simulator. Using the ideas we have developed on this project, she implemented recovery from a number of failure conditions, including loss of traction and loss of steering. The system was able to recognize the failure based on the sequence of irregular sensor inputs, and then it simulated many possible recovery strategies. In almost all cases, the car was able to avoid crashes. In some failure cases, it was even able to continue driving. In parallel, another team of undergraduates modified a remote-control car by adding computing and sensors. We then implemented this failure recovery method on the real robot. Although it was more difficult to produce failure conditions in the lab (such as loss of traction), we were able to show that it could drive stably even in low friction on steep slopes without losing control. This work has not yet been published.

4.2 Long-Term Mission-Planning to Maximize Information Gain

The project fully achieved its goals in the area of mission planning. We demonstrated a capability for a robot to perform route planning online for the purpose of maximizing information gain. Furthermore, we showed that

from even a small set of observations (under 1% of the total terrain), it is possible to reconstruct an entire map with low reconstruction error. These results were reported at RSS 2019 [11] and are currently in submission to IROS 2020 [12]. These results provide a bound on suboptimality. We demonstrated the algorithm's effectiveness on real-world data sets, including NDVI (vegetation health) data and fall foliage colors. These are representative of many other globally observable patterns, such as ocean temperature and salinity.

4.3 Adaptible Specification

In RSS 2018 [1], we demonstrated a quadcopter flying in simulation using landmark-based visual navigation from natural language instructions. In CoRL 2018 [2], we extended the technique to perform planning in addition to control by predicting the probability of passing over each point in the map. At CoRL 2019 [3], we demonstrated transfer of this learned model from simulation to the real quadcopter robot while using very few new training examples in the real environment. We have since extended the method to navigate using unfamiliar landmarks as well by inferring their semantic grounding. That work will be submitted to RSS 2020.

5 Recommendations

Persistent autonomy is a broad challenge. Before we can solve it, we will need to make advances in almost every aspect of the robotics problem. In addition, many approaches will need to be completely rethought.

As a consequence of three years spent working on the topic of persistent autonomy, I have been able to lay out the persistent autonomy problem space in more detail than has been articulated previously. What follows is not a formal roadmap but rather the product of these past three years of active research. The project scratched the surface of a few of these problems. However, solving persistent autonomy is beyond the scope of any one researcher. It will take a movement, and so my objective in recording these ideas is to highlight promising directions in which to move the community. It could make sense to convert what follows into a position paper to be circulated within the robotics and AI communities.

5.1 History of Persistent Autonomy

Before we explore these insights, let us consider first the paradigm under which long-term autonomy has been developed to date. The major success story here is the manufacturing sector, in which robots are programmed in excruciating detail to operate continuously for months at high precision in a totally controlled environment. Most factory robots operate within a cage, whose purpose is to keep people out — not only for safety but also to protect the robot's environment from being disturbed by people. Although people are not physically copresent in the cage, expert engineers monitor the robot's operation and can intervene if needed to repair faults.

When a product design is updated, it can take a week for an expert programmer to reprogram a factory robot. One cause of this inefficiency is that the robot itself provides no assistance. It typically knows nothing about the task it is performing nor the significance of its motions in the task context. Instead, the robot knows only the motions it must perform, and the process engineer is responsible for ensuring that the parts and materials are correctly positioned. For this reason, factory robots are sometimes referred to as "time-shifted teleoperation" rather than true autonomy.

In contrast to robots themselves, some other factory automation mechanisms that control the environment are works of art as much as engineering. For example, vibratory bowl feeders are used in factories to orient fasteners from a bin uniformly so that a robot can blindly grab and install them. The feeder mechanism relies on the randomness produced by vibration; it contains a track with a series of gates that knock out parts that are in the wrong orientation, so that they can recirculate in a new random orientation. Vibratory bowl feeders are a classic and widely-used device, yet no general method for designing a vibratory bowl-feeder exists for an arbitrary part. It is a product of human creativity that defies precise algorithmic description or constructive specification. Intuitionistic approaches to robotics like this are favored for applications that reduce uncertainty. The very unpredictability of this problem makes it hard to explicity specify or design an engineered solution.

We distinguish the open-world persistent autonomy problem from the factory automation problem in two ways. First, the increased passage of time since the "launch" of the system (termination of human monitoring and intervention) increases the opportunity for circumstances or conditions to change. Second, by placing the robot in an uncontrolled environment, there

are more potential sources of change (disturbances) at any one time. Both of these increases in task complexity also require an attendant increase in specification complexity if we are to fully instruct the robot on how to behave in all circumstances. Thus, it is incumbent on the engineer to anticipate many more eventualities when designing and building a persistently autonomous system.

To make matters worse, the aspects of autonomy that are not handled by factory automation but are needed in the open world – that is, the control of uncertainty, like the vibratory bowl feeder – is the part of the process that is hardest to specify a design for. This has been one of the core stumbling blocks of robotics research over the last several decades. Uncertainty is very hard to handle algorithmically. With the state of modern perception, autonomy must distinguish between uncertainty that is intrinsic to the sensors and naturally-occurring uncertainty in the open world.

It would seem that the fundamental problem with the modern approach to building autonomous robots is that it still bears a substantial resemblance to the old time-shifted teleoperation. Engineers try to anticipate every eventuality and specify precisely how the robot should respond. I believe that we have reached the limits of this offline problem-solving approach, in which humans anticipate and specify how a robot should respond. It is natural for an engineer to want to specify how a robot can address every possible scenario. Instead, we should consider moving robotics to an online decision-making approach, recognizing that much more information is available to make a decision in the moment by the robot than in advance by the engineer. That is, the autonomy problem becomes simpler if we can shift more authority to the autonomy.

5.2 Specifications

Much robotics research amounts to an extension the factory automation paradigm. The hallmark of factory automation is precise, engineered behavior. Factory robots follow a design and fabrication process similar to most of the engineered products they assemble. When we design a product, the best engineering practice is to write down a complete, detailed specification for how the product should perform. A specification is a set of requirements that can detail myriad aspects of system performance. There are many types of requirements. Constructive requirements state how the robot should work (e.g. an algorithm). Descriptive requirements give properties of the imple-

mentation, or of the running system, without stating how it is achieved. There are of course many others types of specification as well.

Both the formal product engineering approach and the intuitionistic approach to building automation tacitly assume that you can test it and validate that it works, either empirically or with formal verification methods. Unfortunately, the time required for either approach is exponential in the size/complexity of the specification of the system. Thus, these validation methods do not scale well to the length and scope of general persistent autonomy applications. Solutions to the persistent autonomy problem simply cannot be exhaustively tested. At the same time, the state of formal methods research for program synthesis and verification is not yet mature enough to use on whole systems.

Research Questions. We are left with several major open challenges regarding specifications:

1. How can we write specifications more abstractly for robots to make decisions themselves rather than telling them what decisions to make?

2. How can we validate that such a specification is correct for the innumerable possible situations such a robot may face?

3. How can we engineer systems that will make trustworthy decisions themselves, given these new specifications?

4. How can we build robots that have the situational awareness needed to make decisions competently?

5. How can we validate that an implementation meets such a specification?

5.3 Unanticipated Events

Sometimes unusual events occur that require an appropriate response. Consider a deer that walks in front of a moving car, a sensor that malfunctions, an aircraft that strikes a bird, etc. These events are characterized by their infrequency, discrete nature, enumerability, and the necessity of a response.

Under long-term deployment, unusual events play a different role than they do in laboratory experiments. In a typical laboratory experiment, most failures that arise are of a common enough variety to be reproducible. For

example, a walking robot mispredicts the ground plane where it steps and topples over. Given the limitations of modern autonomous perception, it should not be too hard to reproduce such a failure. Thus, the process of engineering robot systems today tends to mirror the develop-and-test practice of software engineering, in which extensive testing is used to shake out bugs in the system.

Let us consider the set of unusual events that are not seen in testing, which we can call *unanticipated events*. These events still require an appropriate response by the robot, but one of the following applies: they were not written as a constructive specification, they were not implemented on the robot due to an assumption that they would not happen, or they were not validated in testing. When a robot operates in the open world beyond the laboratory, the number and probability of unanticipated events dramatically increases. As operation in the open world without human intervention becomes prolonged, the occurrence of some unanticipated event becomes inevitable. This includes unanticipated events that could be classed as "failures". In this discussion, we stick to the broader term "unanticipated event".

The process of handling an unanticipated event has four steps. Each one is illustrated with an analogous example showing a human pilot responding to an unanticipated event.

1. **detection**: The robot must detect that an unanticipated event is happening, has happened, or is about to happen.
   Ex: The pilot sees a warning light in the cockpit light up.

2. **diagnosis**: The robot must understand what event is happening.
   Ex: The pilot notes that the warning light indicates an engine has stalled.

3. **explanation**: The robot must to explain why the unanticipated event is happening.
   Ex: The pilot notes which warning light is on (engine stall), accounts for the context in which it came on (low altitude), and forms a theory about what the unanticipated event is (foreign object debris, perhaps a bird strike).

4. **response**: The robot must determine, based on the above information, what action to take in response.
   Ex: The pilot makes a plan to circle around and land on the same runway using the one remaining engine.

This template is general enough to respond to any unusual event, although some steps may be no-ops in some situations. The four steps require the robot to possess considerable knowledge about itself, its circumstances, and the actions it is capable of performing. We address the larger implications of this in the following sections. One observation of note here is that unlike the event itself, the response may be either short term (instantaneous) or long term (durable). The difference is whether the response affects future actions that the robot takes.

Durable responses to unanticipated events are a mechanism for adaptation. As circumstances change during long-term autonomy in ways that may be hard to predict, adaptation becomes an essential component of robustness.

We may wish to deprioritize a popular practice within the robotics research community, which is the focus on optimality. This property can be appealing because it tends to cater to theoretical proofs. However, under high-uncertainty conditions, an optimal expected outcome may have an arbitrarily bad worst case outcome, given that the distribution assigns a small probability to that outcome. We would be better off building robots that emphasize avoidance of worst case outcomes while maintaining a "good enough" expected case. In general, strictly optimal performance is not as crucial as bounded suboptimal performance with good average case performance. That is, we would like our robots to perform well on average and live to fight another day.

Case study. One of the best examples to date of attempted persistent autonomy is the various Mars rovers and orbiters. Since the process of landing or entering orbit proceeds in real time, the half-hour latency prevents people on Earth from having situational awareness or intervening to help guide the spacecraft as it appraoches the Martian atmosphere. The Mars Climate Orbiter burned up on approach in 1999 due to a units conversion error between software modules, causing it to approach Mars too rapidly. Had a human pilot been aboard, he or she might have realized that the spacecraft was not slowing down enough to enter orbit and intervened and made a correction. The engineers who built the MCO did not sufficiently anticipate or test for units conformance errors. It would not have been a difficult test to write or run, but this particular failure was hard to anticipate in the traditional develop-and-test mindset since it required the engineers to assume that the specification and/or implementation were wrong despite extensive testing. This failure illustrates the value of automated tools capable of performing formal verification of code. In addition though, some higher-level

judgment must exist on persistently autonomous robots to replace the role that humans play in handling unanticipated situations.

Promising approaches. How can we handle unanticipated events that we cannot plan and test for explicitly, while also specifying nominal behavior precisely? One notion is to adopt a hierarchy of specifications. I will return to this notion shortly. One way to approach the design of a hierarchy for generalizing over a broad class of behaviors is to take inspiration from Maslow's hierarchy of needs. Using this approach, I propose the following priorities, ordered from highest to lowest: homeostasis (steady internal conditions), energy, security, mission. This hierarchy places the robot's priority on self-preservation, with the consequence that some of those more basic needs may interfere with accomplishing the mission. There may be circumstances for certain robots and/or missions where the order could differ. One advantage of the hierarchical needs approach is that it raises the possibility of creating isolation or independence among the layers. This notion bears some relation to Arkin's behavior-based architecture and Brooks's subsumption architecture that were popular in the 1980s and 1990s. It is worth revisiting those as a starting point for a persistently autonomous system.

Another issue deserving more study is abductive inference, which is an under-studied problem, particularly in robotics. Abduction, sometimes called "inference to the best explanation", has been used to solve very complicated problems like crime solving and medical diagnosis. Many of the problems a robot faces that are caused by unanticipated events are actually much simpler than these mentally taxing problems. Such simple inference could possibly have saved the Mars Climate Orbiter when it started to descend too rapidly. Perhaps counterintuitively, easy abductive inference has received far less research attention than the hard cases. It is these simple, "everyday abduction" cases that robots most frequently need in order to adapt to changing conditions. Research on this topic needs to emphasize rapid over precise evaluation. This capability would enable robots to rapidly develop and evaluate multiple hypotheses involving unanticipated events in a short enough time-frame to respond successfully to the event.

Research Questions. Here are several research questions that need to be answered regarding unanticipated events:

1. What capabilities does a robot need to have in order to handle unan-

ticipated events?

2. How can a hierarchical specification be designed to cover what a robot should usefully do in all possible unanticipated events?

3. How can we write specifications that permit needed but unanticipated adaptations?

4. How can we design and validate the interactions between levels in the hierarchy without causing unpredictable emergent behavior?

5.4 Knowledge Representation

Persistent autonomy requires advances in knowledge representation. Although this field is comparatively mature within AI, there are directions particularly relevant for robotics that have been less explored. Robots are concerned about using knowledge to make decisions. Every decision commits to a single action and rejects all other choices. This in turn requires committing to an interpretation of current knowledge under uncertainty. It is not uncommon in certain branches of AI these days to see researchers counting as correct any answer in the top 5 ranked answers. In contrast, in robotics any choice lower-ranked than the first counts for nothing because it is the road not taken. This fact has consequences for both knowledge representation and tasking.

In classic AI research, knowledge is isolated in an abstraction, whereas in robotics we care about knowledge contextualized in the robot's physical world. This has a few implications. First is the open-world problem: how can a reasoning system incorporate new symbols and new knowledge over the lifetime of the system? Second, all knowledge originates as data in the form of sensor signals and control inputs. Thus, before doing any inference, we must first ask: how does knowledge become knowledge? Concepts the robot learns must be grounded in signals (this is the symbol grounding problem). The inverse of this problem is to construct new symbolic knowledge from these signals. Both of these problems must be solved constructively without human supervision and with resilience to noisy data. Thus, knowledge must be tagged with some kind of confidence level in order to perform revision of beliefs. We should explicitly eschew forcing robots to use symbols, labels, or classes that are intuitive to humans, noting that these are not necessarily useful or distinguishable for a robot.

This, in turn, gives rise to the bootstrapping problem. Where does initial knowledge come from? When every observation is novel, then noise and signal appear equally important. There are two traditional approaches to bootstrapping. First, systems may possess a pre-curated ontology. Data within is considered gospel, or is somehow assigned a confidence. Second, robots may be "born" with no knowledge and must learn everything. In both instances, any new information added to the system must be inferred via noisy sensing. Consequently, all information learned by the system is uncertain to some degree and subject to revision. Thus, a robot needs a means to revise previously-learned information. In the first case, this creates two classes of knowledge (one revisable, and one not). In the second case, it can take a very long time to learn basic skills from a blank slate. Previous studies in evolutionary robotics, developmental robotics, and reinforcement learning often produce goofy, undesirable behaviors unless people spend a lot of time tuning reward functions.

Another issue that the system must competently handle is counterfactuals. The system must be able to posit knowledge that it does not know to be true, in order to evaluate what the consequences of this knowledge would be. Planning is an example of counterfactual reasoning that robots routinely face. This problem also arises when evaluating unanticipated events. Abductive inference can be used as a form of counterfactual reasoning to explain mundane or surprising observations that the robot makes.

Beyond all of these requirements, another is the need to organize information for efficient search. Operating in the world creates real-time decision-making deadlines. This in turn places deadlines on knowledge retrieval and inference. Work on cognitive architectures has studied models with different levels of memory differing in recall speed, akin to models of human memory.

Promising approaches. The problem of establishing an ontology and grounding of symbols in parallel has rarely been approached from this direction. The "everyday abduction" technique may be valuable for extracting knowledge from signals. Suppose that many signals over time are projected into a high-dimensional embedding space constructed in a manner that is significant to the physical process being sensed. Then signals whose projections into the embedding space are in proximity would naturally be clustered into a common symbol. For example, a robot manipulator could learn to associate textures by tactile and visual perception of objects, where the clusters might

correspond to smooth, rough, mottled, etc. Extracting these clusters would solve the symbol grounding problem. Advantages to this approach are that all symbols have inherent meaning and that beliefs can be assigned a confidence score based on the nature of the embedding space projection and the clustering that was discovered. It provides a tool to evaluate counterfactuals (since they would have an inherent mapping into sensable properties of the world). Finally, if evidence is later collected that contradicts an earlier belief, then they both can be weighted based on the evidence, and hypotheses can be readily revised or reconstructed from the original data.

A much more basic related question involves the nature of knowledge itself as represented by a robot. Consider a typical predicate, *The door is open.* This phrase evokes a clear meaning to a human reader, even though a robot may struggle to ground concepts like "door" and "open". The expression seems potent to a human engineer not because of the semantic content of the statement itself but rather what it entails. For example, if the door is open, then we can conclude: there exists a place in the room from which I can observe the hallway, there exists a continuous path from the room into the hallway, while I am in the room, my presence can be known by a hypothetical observer in the hallway, and so on. However, each entailment requires quite a bit of additional semantic knowledge beyond what is contained in the statement. This forces us to ask whether it is even useful for a robot to represent knowledge like *The door is open.* It seems imperative if the robot's goal specification is written in terms of that predicate being true, or if the robot must otherwise communicate with humans. In a persistent autonomy context though, it is not obvious that such high level human abstractions are even particularly useful for a robot, given that lower-level abstractions are also available.

Research Questions

1. What is the right level of abstraction at which to represent symbolic knowledge for persistent autonomy?

2. How can the choice of abstraction be guided by the needs of the robot instead of the intuition of the programmer?

3. What representation of confidence most effectively meets the needs of robot inference?

4. When new and old beliefs contradict, how can a robot decide which one overrides the other?

5. How can counterfactual reasoning be kept tractable, given the unbounded possible counterfactuals and explanations?

6. For efficiency, can probable counterfactuals be precomputed, for example from experience?

7. Is there a way to anticipate what knowledge or inference is most likely to be needed rapidly and unexpectedly?

8. When performing abduction on signal clusters, how should the parameters for the embedding space be selected? Are there physical processes that give natural preferred options?

5.5 Tasking

An autonomous robot must task itself by constructing plans to achieve tasks such as communication, delivery, or reconnaissance. Such plans represent decisions that commit the robot to act within the world. The planning problem statement, as conventionally expressed, is to find a sequence of actions that transform the current state in order to make some predicate true. Thus, planning is fundamentally an exercise in counterfactual reasoning. This category includes two types of planning, called task planning and motion planning in robotics. The two are differentiated by the level of granularity and abstraction at which actions and constraints are represented.

In long-term autonomy, a core question is how can a robot know what to do with itself? This is a question of how to write a specification to task the robot. Since a plan is itself a specification, this is really a question of writing meta-specifications.

Let us consider the two traditional robotics planning problems: task planning and motion planning. Motion planning is continuous and geometric in the style of RRT (rapidly-exploring random tree) or PRM (probabilistic roadmap). Traditionally, it requires a precise geometric map of a space inhabited by the robot and uses a kinematic abstraction of robot motion. A solution is expressed in terms of a trajectory in the robot's configuration space. A robot can automatically generate obstacle maps suitable for doing motion planning using geometric sensing, although the goal is typically in

service of some higher objective that is abstracted away. Task planning is symbolic and abstract, in the style of STRIPS (Stanford Research Institute Problem Solver) or PDDL (problem domain definition language). This type of planning operates in a Boolean predicate space requiring human intuition to devise a coherent abstraction of a problem. Such planning problems generally cannot be automatically posed without human help because the abstraction incorporates substantial non-geometric information. For example, a pair of blocks on the table are stackable, but the stackable affordance is hard to detect geometrically. This affordance requires many assumptions: that the blocks are not attached to the table, that they are rigid enough to grasp, that they are light enough to pick up, that their coefficient of friction is suitable for grasping, and so on.

The two planning problems are quite distinct. They are effectively projections of the total planning problem onto orthogonal subspaces. Solutions to the two problems are typically complementary, but in practice it is quite hard to combine them. My group has made recent progress towards solving the two problems jointly [14]. The challenge inherent in solving these two problems suggests that we may gain some leverage from reposing the problem in a manner that is more suitable for autonomous specification and solution.

All knowledge on which the robot acts has uncertainty. Therefore, it follows that plans must be generated in a manner that mitigates uncertainty. However, this notion contrasts with the conventional expression of the motion planning problem as the search for a deterministic plan. Under uncertainty, a new expression of the planning problem is needed. Types of uncertainty that affect planning include: current state, goal, constraints, and effects of actions on state. When making decisions under uncertainty, it is easy to overcommit to a particular course of future action. Overcommitment here means that at the future time when the robot must implement the decision by taking the planned action, better information is available that would lead to better choice being made at that moment. Thus, an important aspect of dealing with uncertainty involves trading off decisions now versus deferred decisions. The robot cannot simply defer all future decisions, however, because the reason for doing planning is to choose an action now that leads to the best eventual outcome, and the robot cannot know which action leads to the goal without considering the entire sequence of future actions. Techniques like interleaved planning and execution have been developed to address this problem. The idea is that a robot computes an entire action sequence, executes the first step of it, and then replans from the new state from scratch.

I believe more efficient approaches are possible by planning abstract actions and then gradually refining them.

Abstraction is really a key issue that must be further explored. As I mentioned previously a few notions of levels of abstraction are baked into the specification of planning problems (symbolic task planning, and geometric motion planning). For persistent autonomy, we are acquiring knowledge autonomously and building the specification of the problem autonomously. I think it is a mistake to insist that the interpretations of the problem that the robot solved must be what is intuitive to a human. We need to revisit at what levels of abstraction planning is most appropriately performed. These choices should reflect both the nature of uncertainty and the forms of knowledge that a robot readily learns. In particular, the symbolic task specification may be too abstract – at least until a robot can itself derive symbols at the appropriate level of abstraction.

As previously discussed, there may be multiple levels of mission goals. This fact may result in multiple levels of plan executing simultaneously at varying priority. For example, bringing a drone back home may be higher priority than getting certain photos, or it may be lower. This feeds into risk management, which is a more natural place for human specifications to be made than particular geographic coordinates or abstract symbolic properties.

Promising approaches. The task and motion planning problem is currently defined in a way that is intuitive for people to specify, but difficult for a robot to solve. The core motivation for the use of task planning is that the robot may discover plans to achieve goals requiring many sequential actions, provided that the actions themselves may be abstracted and then readily generated on demand via a motion planner. This approach to planning is but one on a spectrum of approaches that vary in abstraction. We need to redefine the spectrum of graduated levels of abstraction that will enable a robot to slide along the spectrum efficiently, from accurate, physics-based simulation of contact with the world to geometric motion and higher levels of abstraction.

If we consider multiple abstractions at the levels of physics/dynamic, geometry/kinematic, and pure symbolic, then we must also consider types of uncertainty applied to each of these. Present approaches to uncertainty are clustered into a few categories. Deterministic approaches assume one possible outcome of all random processes and proceed to generate a single,

precise solution accordingly. If for any reason the precise solution becomes infeasible later, the entire plan must be thrown out and computed from scratch. Stochastic approaches attempt to find the plan that will succeed with maximum probability given current knowledge. This solution may itself be expressed probabilistically, or it may be a single, precise solution as in the deterministic approach. One popular framework for modeling planning under uncertainty is POMDP (partially-observable Markov decision process), which considers the robot as playing a game against nature and trying to act optimally at each step. These approaches are all expensive to compute because they share a common flaw of needlessly overcommitting to a solution.

I believe an alternative approach can perform the same computation at a lower computational budget by finding a least-commitment plan. Such a plan provides at each execution stage a set of actions that would result in progress if executed. The planner then leaves to the controller the choice of which specific action to execute. The virtue of this approach is that it defers as many decisions as possible to the time when the most information is available. A side benefit is that if the controller never needs to make a particular decision, then no time is spent precisely computing it. Another benefit is that the approach is more robust to disturbances. After a disturbance, the robot may find itself in a new state from which an approximate plan has already been computed, so handling the disturbance may require no special effort. For example, in the realm of geometric motion planning, a least-commitment plan specification could be expressed topologically via a deformation equivalence relation similar to homotopy. Such a notion, grounded in the perceptions of an autonomous robot, may provide the basis for a spectrum of levels of abstraction for planning.

Research Questions.

1. How can least-commitment plans be most generally represented?

2. How can we let robots automatically select appropriate levels of abstraction for a given task?

3. How can uncertainty be represented without the overhead of probability (e.g. performing endless integrals of minuscule probabilities)?

4. Given a language grounded in a robot's percepts, how can we specify tasks at an appropriate level of abstraction?

5. Given a language grounded in a robot's percepts, how can unobserved attributes, such as a future goal, be understood?

5.6 Systems Engineering Issues

The robotics field has paid surprisingly little attention to many systems problems [13]. Robots are fundamentally different from any other engineered product, for the reasons discussed in Section 5.2.

One characteristic of robots is that they are complex systems with many interacting components. Aircraft and spacecraft share this property, and both are giant enterprises that take years to design, build, and test. Validation is a very substantial component, including both testing and verification. At the end of this process, these systems are considered highly reliable. Although major incidents like the recent 737 MAX crashes are sensational, they are extremely rare. This test process can be as reliable as it is because it assumes there is a human making decisions at the time when a rare event occurs. A consequence of this assumption is that systems can get away with having limited situational awareness. In addition to evaluating the system as a whole, individual components are validated in isolation. Such component tests are meaningful because the overall system design prevents emergent behaviors due to interactions among components. In contrast, the possibility of multiple rare events interacting is harder to test. Some independence between rare events can be assumed since a trained human will be able handle such interactions. For persistent autonomy, the costs of validation increase dramatically since the high-level decision maker is part of the system under test.

Robot systems are often built with a laboratory demonstration in mind, which can lead to certain decisions that limit later scalability. Many forms of scalability matter, including run time, number of robots coordinating, number of tasks performed, size of the environment, and so on. System development needs to emphasize not limiting scalability needlessly, and testing each form of scalability as much as possible.

Robot engineering could be characterized as the process of making assumptions, abstractions, approximations, and generalizations about the world that simplify the problem enough to be tractable to solve. The trouble is that each of these choices is a simplification that will be wrong some fraction of the time. A good architecture should isolate the system against the risks inherent in the simplifications being wrong because the alternative is

failure. Unfortunately, current robot architectures have many interdependencies among components.

Decisions made by engineers at design time have a profound effect on how a robot is capable of generalizing. Robot architectures are typically modular, consisting of components whose interconnections form information flow graphs. In a simple example, a sensing module connects to a mapping and localization module, which connects to a planning module and finally a control module. This sequence describes a "planning pipeline". The design-time choices about which modules to include and how they connect create tacit restrictions on what conditions can be handled by the robot at runtime. Such communication flow graphs allow us to analyze risk at the component level by noting critical points through which all information must flow. Such critical points are single points of failure if a component misbehaves either due to a bug or an unmet latent assumption. Since engineers design systems for components to work correctly, single points of failure are not guarded against in robotics, and so they are everywhere. Redundancy is one possible answer, although we must consider what benefit redundancy adds to a system. Simply replicating an existing component (such as was done on the space shuttle) guards against hardware failure but not bugs or latent assumptions, which are arguably the bigger problem. An alternative approach, then, might be to diversify the methodological approach among the redundant components.

Perhaps redundant modules can incorporate alternative implementations. For example, we can imagine generating competing estimates using both a particle filter and a Kalman filter, or complementary forms of map. This approach gives rise to multiple different solutions to every problem. Traditional architectures avoid needing to choose between contradictory estimates or actions by including only one of each component in the pipeline. This serial architecture does not truly avoid making these decisions but rather locks in certain choices at design time with zero situational awareness, thus limiting the robot's situational awareness and flexibility at runtime.

Fundamentally, robots are decision systems, so no matter how many sources of input there are, the robot must commit to a single control (output) at one time. If we are to parallelize an architecture by adding competing modules, then there are also decision problems concerning how much to trust the various information sources. Thus, there are many combinations by which modules can provide input to a decision problem. These combinatorics make testing a nightmare, which is one reason why this practice is not generally followed. The validation problem will need to be addressed in order

to leverage novel architectures that exploit parallelism and redundancy.

Promising approaches. It would be worthwhile to investigate wholly new architectures for building robot systems that maximize runtime flexibility. Even within the current serial pipeline, modules tend to overcommit to decisions too early. For example, robot motion planners tend to output a single geometric trajectory for the controller to follow. This choice is somewhat arbitrary (there are many similar paths that would have been equivalently good choices) and sets up the controller to fail in the sense that all controllers have some degree of path-following error. In the case of motion planners, a preferred strategy would be to emit as a solution a continuous set of paths, such as a homotopy class or neighborhood of paths. A controller could then follow the class of paths, like driving in a tunnel, with the knowledge that any path is allowable provided it stays inside the boundary. By deferring the choice to the controller, we allow for the most current information to be factored into robot's decision making. Similar principles apply throughout the planning pipeline, where there are opportunities to defer decisions by representing partial computations or satisficing sets rather than arbitrarily picking one solution among many.

Research Questions. Here are some research questions regarding system engineering:

1. What fundamental assumptions about the way that robots are architected can be reexamined?

2. How can architecture be modified to mitigate the effects of assumptions that don't hold true?

3. How can architecture be modified to maximize runtime decision-making flexibility?

4. How can new architectures with greater redundancy be validated?

6 Conclusions

This project touched on a few small aspects of persistent autonomy. Working in these few areas highlighted a number of major research problems that

must be solved on the way to persistently autonomous robots. Like the QWERTY keyboard, which long outlasted its purpose in preventing mechanical jams, many vestiges of the past survive as unchallenged assumptions in robotics. The biggest obstacle may then be the body of assumptions governing how robots are architected, designed, built, and programmed. Consequently, many of these problems are going to require substantial investments in time and resources to build sustainable progress that could lead the robotics community in new directions. Although going back to basics may not seem like progress, it is necessary in order to get the community focused on solving the right problems that will lead to reliable persistent autonomy.

References

[1] Valts Blukis, Nataly Brukhim, Andrew Bennett, Ross A. Knepper, and Yoav Artzi. Following high-level navigation instructions on a simulated quadcopter with imitation learning. In *Proceedings of the Robotics Science and Systems Conference*, Pittsburgh, USA, June 2018.

[2] Valts Blukis, Dipendra Misra, Ross A. Knepper, and Yoav Artzi. Mapping navigation instructions to actions with position visitation prediction. In *Proceedings of the Conference on Robot Learning*, Zurich, Switzerland, October 2018.

[3] Valts Blukis, Yannick Terme, Eyvind Niklasson, Ross A. Knepper, and Yoav Artzi. Learning to map natural language instructions to physical quadcopter control using simulated flight. In *Proceedings of the Conference on Robot Learning*, Osaka, Japan, October 2019.

[4] B. Roy Frieden. *Science from Fisher Information: A Unification*. Cambridge University Press, 2004. doi: 10.1017/CBO9780511616907.

[5] Bruce Golden, Lawrence Bodin, T Doyle, and W Stewart Jr. Approximate traveling salesman algorithms. *Operations research*, 28(3-part-ii): 694–711, 1980.

[6] Abolfazl Hashemi, Mahsa Ghasemi, Haris Vikalo, and Ufuk Topcu. A randomized greedy algorithm for near-optimal sensor scheduling in large-scale sensor networks. In *2018 Annual American Control Conference (ACC)*, pages 1027–1032. IEEE, 2018.

[7] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[8] Christoforos Mavrogiannis, Valts Blukis, and Ross A. Knepper. Inferring strategies of avoidance: Towards socially competent navigation in human environments. In *Workshop on Mathematical Models, Algorithms, and Human-Robot Interaction*, Proceedings of the Robotics Science and Systems Conference, July 2017.

[9] Christoforos I. Mavrogiannis, Valts Blukis, and Ross A. Knepper. Socially competent navigation planning by deep learning of multi-agent path topologies. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vancouver, Canada, September 2017.

[10] European Comission Seventh Framework Programme. Pandora project: Persistent autonomy through learning, adaptation, observation and replanning. URL https://cordis.europa.eu/project/id/288273.

[11] Elizabeth Ricci and Ross Knepper. A bounded suboptimal environmental monitoring algorithm. In *Robotics: Science and Systems Workshop on Informative Path Planning and Adaptive Sampling*, Freiburg, Germany, June 2019.

[12] Elizabeth A. Ricci, Madeleine Udell, and Ross A. Knepper. A modeling, prediction, and information-theoretic approach for environment monitoring. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, in submission 2020.

[13] Soham Sankaran and Ross A. Knepper. Usability squared: Principles for doing good systems research in robotics. In *Robotics: Science and Systems Workshop on Cloud and Fog Robotics in the Age of Deep Robot Learning*, Freiburg, Germany, June 2019.

[14] Wil Thomason and Ross A. Knepper. A unified sampling-based approach to integrated task and motion planning. In *Proceedings of the International Symposium on Robotics Research*, Hanoi, Vietnam, October 2019.

[15] Madeleine Udell, Corinne Horn, Reza Zadeh, Stephen Boyd, et al. Generalized low rank models. *Foundations and Trends® in Machine Learning*, 9(1):1–118, 2016.

[16] Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. OBOE: Collaborative filtering for AutoML initialization. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2019. URL https://www.kdd.org/kdd2019/accepted-papers/view/oboe-collaborative-filtering-for-automl-model-selection.

[17] Haifeng Zhang and Yevgeniy Vorobeychik. Submodular optimization with routing constraints. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.