

US Patent & Trademark Office

Patent Public Search | Text View

United States Patent Application Publication

20250259031

Kind Code

A1

Publication Date

August 14, 2025

Inventor(s)

OLSON; Edwin et al.

Method And Apparatus For Constructing Informative Outcomes To Guide Multi-Policy Decision Making

Abstract

In Multi-Policy Decision-Making (MPDM), many computationally-expensive forward simulations are performed in order to predict the performance of a set of candidate policies. In risk-aware formulations of MPDM, only the worst outcomes affect the decision making process, and efficiently finding these influential outcomes becomes the core challenge. Recently, stochastic gradient optimization algorithms, using a heuristic function, were shown to be significantly superior to random sampling. In this disclosure, it was shown that accurate gradients can be computed-even through a complex forward simulation-using approaches similar to those in deep networks. The proposed approach finds influential outcomes more reliably, and is faster than earlier methods, allowing one to evaluate more policies while simultaneously eliminating the need to design an easily-differentiable heuristic function.

Inventors: OLSON; Edwin (Ann Arbor, MI), Mehta; Dhanvin H. (Ann Arbor, MI), Ferrer; Gonzalo (Ann Arbor, MI)

Applicant: The Regents of The University of Michigan (Ann Arbor, MI)

Family ID: 1000008571973

Assignee: The Regents of The University of Michigan (Ann Arbor, MI)

Appl. No.: 19/170316

Filed: April 04, 2025

Related U.S. Application Data

parent US continuation 18653211 20240502 parent-grant-document US 12299554 child US 19170316

parent US continuation 18196897 20230512 parent-grant-document US 12001934 child US 18653211

parent US continuation 17371221 20210709 parent-grant-document US 11681896 child US

18196897

parent US continuation 15923577 20180316 parent-grant-document US 11087200 child US 17371221

us-provisional-application US 62472734 20170317

Publication Classification

Int. Cl.: **G06N3/02** (20060101); **G06N3/008** (20230101); **G06N3/084** (20230101); **G06N7/01** (20230101); **H04N1/00** (20060101)

U.S. Cl.:

CPC **G06N3/02** (20130101); **G06N3/008** (20130101); **G06N3/084** (20130101); **G06N7/01** (20230101); **H04N1/00002** (20130101);

Background/Summary

CROSS-REFERENCE TO RELATED APPLICATIONS [0001] This application claims the benefit of U.S. Provisional Application No. 62/472,734, filed on Mar. 17, 2017. The entire disclosure of the above application is incorporated herein by reference.

FIELD

[0003] The present disclosure relates to a method for multi-policy decision making of an object moving through an environment.

BACKGROUND

[0004] Autonomously controlled objects, such as a robot or vehicle, need to consider their surrounding environment in order to make decisions regarding which direction to pursue. To do this, the controlled object considers all possible scenarios, directions, and outcomes for itself and the objects around them. However, randomly considering each possible scenario is not only computationally expensive, but it is also inefficient as it does not define a method for identifying scenarios where the controlled object may experience collisions or other high-cost occurrences.

[0005] The background description provided here is for the purpose of generally presenting the context of the disclosure. Work of the presently named inventors, to the extent it is described in this background section, as well as aspects of the description that may not otherwise qualify as prior art at the time of filing, are neither expressly nor impliedly admitted as prior art against the present disclosure.

SUMMARY

[0006] This section provides a general summary of the disclosure, and is not a comprehensive disclosure of its full scope or all of its features.

[0007] A computer-implemented method is presented for issuing a command to a controlled object in an environment. A determination is made for each policy from a set of policies, where a policy specifies the command for the controlled object to implement and the command directly affects motion of the controlled object. As a starting point, a state estimate is received for each of one or more monitored objects and the controlled object, where each state estimate includes state elements, and the state elements are indicative of a position of the respective object and a velocity of the respective object.

[0008] Seed states are generated for each of the one or more monitored objects and the controlled object. Movement of the one or more monitored objects and the controlled object is then simulated using the seed states, where the simulation includes the controlled object executing the respective

policy. An outcome of the simulation with the seed states is quantified by assigning a cost to outcome of the simulation with the seed states and determining a probability associated with the seed states.

[0009] Seed states are perturbed (e.g, using backpropagation) and movement of the one or more monitored objects and the controlled object objects is simulated using the perturbed inputs. Simulating movement of the one or more monitored objects and the controlled objects with different perturbed inputs is repeated until a predetermined condition is met, thereby generating a plurality of perturbed outcomes.

[0010] For each policy, a policy score is determined for the respective policy, where the policy score correlates to the perturbed outcome having highest value amongst the plurality of perturbed outcomes for the respective policy. A given policy is then selected from the set of policies, where the given policy has most benign outcome amongst the policies in the set of policies. Lastly, a command is issued to the controlled object in accordance with the given policy.

[0011] Simulating movement of the one or more monitored objects, includes representing trajectory of an object using a differentiable function. In one embodiment, the trajectory of an object is presented by recursively applying a transition function over a series of time steps, where the transition function is defined such that objects are repelled by other agents and attracted towards a goal in accordance with a social force model. In addition, perturbed seed states are determined by iteratively computing gradient for each time step in the series of time steps with respect to the perturbed seed states.

[0012] In some embodiments, the given policy avoids a set of undesired outcomes, and the set of undesired outcomes includes at least one of (i) a collision between the controlled object and one of the one or more monitored objects; and (ii) the controlled object being within a predetermined distance of the one or more monitored objects.

[0013] The cost is determined using a blame metric, where the blame metric is a function of a distance between the controlled object and one of the one or more monitored objects and the velocity of the controlled object. The cost also accounts for the progress toward the goal of the controlled object.

[0014] In some embodiments, the set of policies includes at least one of the following commands for the controlled object to: (i) change a trajectory to follow one of the one or more monitored objects; (ii) remain in the same position; (iii) move forward; (iv) decelerate; and (v) accelerate.

[0015] Further areas of applicability will become apparent from the description provided herein. The description and specific examples in this summary are intended for purposes of illustration only and are not intended to limit the scope of the present disclosure.

Description

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] The present disclosure will become more fully understood from the detailed description and the accompanying drawings.

[0017] FIG. 1 is a block diagram of an embodiment of a multi-policy decision making (MPDM) apparatus;

[0018] FIG. 2 is a diagram of an environment where a controlled object would institute MPDM; and

[0019] FIG. 3 is a flowchart of an embodiment of the MPDM apparatus.

[0020] FIG. 4 is a diagram of a deep network representation for a cost function. The initial configuration x_0 propagates through several layers, each representing the transition function T . The output of layer t determines a cost. The cost function accumulates costs calculated at each time-step along the forward simulated trajectory.

[0021] FIG. 5 is a block diagram of an example transition function. Block diagram of the transition function. At each time-step, an agent i (in this case, the robot) is repelled by other agents ($f_{sub.rep.sup.j}$) and attracted towards its sub-goal $g_{sub.sub}$ in accordance to the Social Force Model (SFM). Pedestrians are modeled using the HSFM model where the social force acts as a control input for the Human Locomotion Model. The robot is modeled like a unicycle and the social force $f_{sub.net.sup.r}$ is transformed into a compliant reference signal ($v_{sub.ref}$, $w_{sub.ref}$) for a lower-level velocity controller.

[0022] FIGS. 6A and 6B are diagrams illustrating backpropagation. In FIG. 6A, the forward propagated outcome of the sampled initial configuration is not discouraging for the robot as it does not inconvenience either agent. For agents $i=\{1,2\}$, the computed gradients $\nabla_{sub.x.sub.0.sub.i} \ln(C(X))$ (Blue) drive the agents towards configurations where the robot would inconvenience them under its current policy while $\nabla_{sub.x.sub.0.sub.i} \ln(P(x))$ (Green) drive them to more likely configurations. The agents can be simultaneously updated resulting in a more influential configuration as seen in FIG. 6B.

[0023] FIG. 7 is a graph depicting degradation of stochastic gradient ascent in crowded scenarios. For each algorithm, estimate the mean and standard error of the number of iterations (forward simulations) taken to discover an influential outcome varying the number of agents in the robot's vicinity, and thereby the dimensionality of the search space. The lower the slope, the better, more robust the algorithm to complex scenarios with high-dimensional search spaces. Random sampling, as expected, requires many samples even in simpler configurations. SGA cannot find influential outcomes efficiently in complex scenarios with multiple agents, scaling so poorly that for more than six agents it performs worse than random sampling. BP is able to find those adverse outcomes even for crowded scenarios with 8 people.

[0024] FIG. 8 is a graph comparing experimental results of the proposed method to the stochastic gradient ascent approach. BP can evaluate 10 policies reliably in real-time, while SGA cannot. Compare the performance of various algorithms on 6 hours of navigation in the simulated environment. Measure the Time Stopped for every goal reached as well as the Blame per meter traveled by the robot. For each algorithm, use bootstrap sampling to estimate the mean and standard error for these metrics, represented by the axes of an ellipse. Lower the Blame or Time Stopped, the better. Run the simulator in real-time allowing a planning time $t_{sub.p}=0.3$ s. Although SGA can evaluate the smaller policy set reliably in real-time, the lack of options results in frequent Stopping. Unfortunately, SGA cannot evaluate a larger policy set of 10 policies reliably and accumulates large Blame. Since BP can evaluate the larger policy set more quickly and reliably than SGA, the robot navigates safely (low Blame) in real-time without Stopping unnecessarily. Upon slowing down the simulator (three times slower than real-time) to allow an unrealistic planning time of $t_{sub.p}=1$ s, one observes that SGA with 10 policies is able to drastically reduce Blame. However, even then BP outperforms SGA.

[0025] FIG. 9 is a graph depicting real-work data from three repeatable experiments. Real-world data was collected from three repeatable experiments represented by different symbols 1) pedestrians crossing orthogonal to the robot's trajectory (+), 2) pedestrians crossing the robot's path obliquely at 45 degrees (A) and 3) pedestrians walking slowly in front of the robot (star). Measure the Time Stopped for every goal reached as well as the Blame per meter traveled by the robot accumulated by inconveniencing pedestrians. Lower the Time Stopped and Blame, the better. The proposed approach (green) can evaluate more policies in real-time than earlier possible. With more candidate policies, the robot can find good policies and can navigate safely without stopping unnecessarily.

[0026] Corresponding reference numerals indicate corresponding parts throughout the several views of the drawings.

DETAILED DESCRIPTION

[0027] In a MPDM framework, a policy of a controlled object is selected by sampling from the

distribution of current states, predicting future outcomes through forward simulation, and selecting the policy with the most favorable population of outcomes. Selecting the best policy depends on sampling initial conditions with influential outcomes.

[0028] The controlled object has a discrete number of policies to choose from in an environment. The controlled object may be a vehicle, a robot, or any other autonomous object that is configured to move through an environment. Instead of selecting a particular trajectory, the controlled object can choose from a set of policies that provide a rulebook for the controlled object to follow. For example, one of the policies may be to follow a monitored object in the environment. When the controlled object is instructed to follow the particular monitored object, the controlled object's trajectory is not predetermined but is dependent on the particular monitored object.

[0029] The model of the environment consists of static obstacles (e.g., walls or doors) and a set of freely moving dynamic agents, assumed to be people.

[0030] The robot maintains estimates of the states of observable agents. The $x_{sub.i} \in X_{sub.i}$ for agent i (including the robot) consists of its position $p_{sub.i}$, velocity $v_{sub.i}$, and a goal point $g_{sub.i}$.

$$[00001] \quad x_i = [p_i, v_i, g_i]^T \quad (1)$$

where each of $p_{sub.i}$, $v_{sub.i}$, $g_{sub.i}$ are two-dimensional vectors. The motion of agents is modeled according to a simple dynamics model in which acceleration, integrated over time, results in a velocity. The force, and hence the acceleration is computed using a potential field method that incorporates the effects of obstacles and a goal point.

[0031] Let N be the number of agents including the robot. The joint state space of the system is $\chi = \chi_{sub.1} \times \chi_{sub.2} \times \dots \times \chi_{sub.N}$. The collective state $x(t) \in \chi$ includes the robot state plus all the agents visible to the robot at time t .

[0032] Our observation model $P(z|x)$ is assumed to be Markovian, where the joint observations z are the pedestrians' positions. For each pedestrian, the goal $g_{sub.i}$ is not directly observable through z . It is assumed to be one of a small set of salient points and is estimated using a naive Bayes Classifier. For the robot, the goal g , is provided by a higher level planner.

[0033] The agent dynamics are defined by the following differential constraints:

$$[00002] \quad \dot{x}_i = [v_i, a_i, 0]^T, \quad (2)$$

The action $a_{sub.i} \in A_{sub.i}$ corresponds to the acceleration governing the system dynamics and is determined by the policy i followed by the agent.

[0034] The transition function maps a given state $a_{sub.i}$ to a new state $T: \chi_{sub.i} \times \text{custom-character.sub.i} \rightarrow \text{custom-characterX.sub.i}$. Thus, the corresponding transition equation is expressed as

$$[00003] \quad T(x_i, a_i) = x_i(t + \Delta t) = x_i(t) + \int_t^{t+\Delta t} \dot{x}_i(r, a_i) dr \quad (3)$$

[0035] In an example embodiment, a discrete set of high-level closed-loop policies is presented.

$$[00004] \quad \mathcal{A} = \{\text{Go - Solo}, \text{Follow}_j, \text{Stop}\}, \quad (4)$$

where $\text{Follow}_{sub.j}$ refers to the policy of following agent j . A robot in an environment with 10 observable agents has a total of 12 candidate policies. Each policy maps a joint state of the system to an action via a potential field $\xi_{sub.i} \in \Xi: \chi \rightarrow \text{custom-characterA.sub.i}$.

[0036] An agent executing the Go-Solo policy treats all other agents as obstacles and uses a potential field based on the Social Force Model (SFM) to guide it towards its goal. Let $e_{sub.pi.fwdarw.g.sub.i}$ be the unit vector towards the goal from the agent i . The attractive force acting on the agent is given by:

$$[00005] \quad f_i^{\text{attr}}(x) = k_{gs} e_{i.fwdarw.g_i} \quad (5)$$

The interactions with other agents in the scene are modeled based on the SFM:

$$[00006] \int_{i,j}^{\text{int}} (x) = a_p e^{-d_{i,j}/b_p} \cdot \text{Math. } e_{j \cdot \text{fwdarw. } i} \quad (6)$$

where $\{p_1, b_{\text{sub.p}}\}$ are the SFM parameters for people, $e_{\text{sub.j.fwdarw.i}}$ is the unit vector from j to i and $d_{\text{sub.i,j}}$ is the distance between them scaled by an anisotropic factor.

[0037] Similarly, each obstacle $o \in O$ in the neighborhood of the agent exerts a repulsive force $\int_{\text{sub.i,o.sup.obs}}(x)$ on agent i according to different SFM parameters $\{a_{\text{sub.o}}, b_{\text{sub.o}}\}$,

$$[00007] \int_{i,o}^{\text{obs}} (x) = a_o e^{-d_{i,o}/b_o} \cdot \text{Math. } e_{o \cdot \text{fwdarw. } i} \quad (7)$$

[0038] The resultant force is a summation of all the forces described above:

$$[00008] f_i(x) = f_i^{\text{attr}}(x) + \text{Math. } \int_{j \neq i}^{\text{int}} + \text{Math. } \int_{o \in O}^{\text{obs}} \quad (8)$$

[0039] The action governing the system propagation (2) is calculated as $a_{\text{sub.i}} = f_{\text{sub.i}}$ (without loss of generality, assume unit mass). In one example, it is assumed that all agents besides the robot always use this Go-Solo policy. In other examples, the policy of the agents are estimated by the robot.

[0040] In addition to the Go-Solo policy, the robot can use the Follow policy to deal with certain situations. The intuition is that in a crowd, the robot may choose to Follow another person sacrificing speed but delegating the task of finding a path to a human. Following could also be more suitable than overtaking a person in a cluttered scenario as it allows the robot to Progress towards its goal without disturbing other agents (low Force). A reactive Follow policy is proposed, making minor modifications to the Go-Solo policy.

[0041] According to the Follow policy, the robot r chooses to follow another agent, the leader, denoted by l . In this case, the robot is attracted to the leader rather than the goal. Let $e_{\text{sub.p.sub.r.fwdarw.p.sub.l}}$, be the unit vector from the robot's position to the leader's position. The attractive force

$$[00009] f_r^{\text{attr}}(x) = k_f e_{p_r \cdot \text{fwdarw. } p_l}, \quad (9)$$

steers the robot trajectory towards the leader. The other agents and obstacles continue to repel the robot as described in (8),

[0042] The last of the policies available to the robot is the Stop policy, where the robot decelerates until it comes to a complete stop, according to the following force

$$[00010] f_r(x) = -f_{\text{max}} e_{y_r}, \quad (10)$$

where $e_{\text{sub.v.sub.r}}$ is the unit vector in the direction of the robot's velocity. It is understood that other types of policies fall within the scope of this disclosure.

[0043] To prevent the controlled object from causing disruptions in the environment, it is necessary to discover likely catastrophic outcomes, such as collisions, occurring when each policy is being implemented. Previously, random sampling has been used to predict potential outcomes. Random sampling is conducted based on a position of the controlled object and the monitored objects.

However, discovering likely catastrophic outcomes through random sampling is computationally expensive as it is likely to miss high-cost events, even if they are reasonably probable, because of the scarcity of such configurations in the environment. That is, catastrophic or high-cost events may have a high probability density on an individual basis, but when evaluating all occurrences of these high cost events, the total probability mass is low.

[0044] Therefore, instead of random sampling, influential outcomes can be identified by perturbing configurations of the controlled and monitored objects iteratively towards those influential outcomes. By locating influential outcomes, the controlled object identifies more high-cost events or worst case scenarios. Then the controlled object may select the policy whose population of discovered likely high-cost outcomes is more benign or desirable. That is, the controlled object is evaluating the quality of each decision available and selecting the decision with the fewest bad

outcomes.

[0045] Referring to FIG. 1, a block diagram of an embodiment of a MPDM apparatus on a controlled object **100** is illustrated. The controlled object **100** includes a controller **104** and a data store **108**. The controller **104** generally includes a perception module **112**, a seed state generator **116**, a simulator **120**, a perturbing module **124**, and an outcome quantifier **128**, and the data store **108** includes probability distributions **132** and policies **136**. In this application, including the definitions below, the term “module” or the term “controller” may be replaced with the term “circuit.” The term “module” may refer to, be part of, or include: an Application Specific Integrated Circuit (ASIC); a digital, analog, or mixed analog/digital discrete circuit; a digital, analog, or mixed analog/digital integrated circuit; a combinational logic circuit; a field programmable gate array (FPGA); a processor circuit (shared, dedicated, or group) that executes code; a memory circuit (shared, dedicated, or group) that stores code executed by the processor circuit; other suitable hardware components that provide the described functionality; or a combination of some or all of the above, such as in a system-on-chip.

[0046] The controller **104** is configured to issue a command to the controlled object **100**. The command is determined by the policy **136** the controlled object **100** has determined to be the best policy based on the current state of the controlled object **100** and the states of a plurality of monitored objects existing in an environment. For example, the environment may be a hallway, a large room, or any other area with static obstacles. The environment may also consist of freely moving dynamic agents known as monitored objects. The monitored objects include those objects, agents, vehicles, humans, etc. that move freely and independent of the controlled object **100**. It is possible to track monitored objects and estimate their state. The perception module **112** estimates a distribution over the state of each monitored object within a certain vicinity of the controlled object **100**. It is impossible to accurately predict the future motion of the monitored objects. Given a state for the monitored objects, through the seed state generator **116** or the perturbing module **124**, the simulator **120** predicts an outcome. It is envisioned that the objects monitored by the MPDM apparatus can account for stationary and/or moving objects.

[0047] The perception module **112** locates the monitored objects in the environment and maintains an estimate of a state for each of the monitored objects. The state for each monitored objects includes a value for the position, speed, and an inferred goal (or policy). The state information for each monitored object may be defined with respect to the controlled object **100**. For example, once identified, one monitored object may be a certain distance from the controlled object **100** and travelling at a certain speed with respect to the controlled object **100**. The motion of the monitored objects is modeled according to a simple dynamics model in which acceleration, integrated over time, results in a velocity. The force, and hence the acceleration, is computed using a potential field method that incorporates the effects of obstacles and the inferred goal point. Other types of motion models are also contemplated by this disclosure.

[0048] The controlled object **100** maintains the estimate of the monitored object's perceived location and speed through the perception module **112**. The perception module **112** identifies and evaluates the array of state values for each monitored object for use during forward simulations. The perceived state values provide a configuration of the environment as a function of time. For the controlled object **100** to decide which policy **136** is best to avoid any catastrophic or high-cost events, the controlled object **100** determines, in real-time, out of all the possible trajectory iterations based on the initial configuration, which policy **136** is least likely to result in a high-cost event, such as a collision. Further details regarding determining the initial configuration of the environment is discussed in U.S. patent application Ser. Nos. 14/814,766, 14/814,856, and 14/814,630, all filed Jul. 31, 2015 and each incorporated by reference in their entirety.

[0049] For each monitored object, the inferred goal is not directly observable through the positions of the monitored objects. It is assumed to be one of a small set of salient points and is estimated using a naive Bayes Classifier. For the controlled object **100**, the goal is provided by a higher level

planner.

[0050] The seed state generator **116** receives the states for each of the monitored objects from the perception module **112**. The seed state generator **116** determines the initial sampling states. For example, based on the initial configuration estimated by the perception module **112**, the seed state generator **116** determines the state values at which to begin the forward simulations based on the estimated position of all the perceived monitored objects. The seed state generator **116** samples an initial configuration from the environment based on the state of each object.

[0051] In alternative applications, Monte Carlo sampling from the posterior distribution of the states of all objects was used to approximate an expected cost for a simulation. To ensure that high-cost events are recognized and not missed during random sampling, the seed state generator **116** identifies influential outcomes. In other words, influential or high-cost outcomes are explicitly searched for and simulated as high-cost outcomes more strongly influence decision-making of the controlled object **100**. For instance, as described in more detail below, state elements (e.g., position, speed, etc.) of the current state of each object may be perturbed while sampling to find high value outcomes (i.e., the product of the probability and the cost). Instead of random sampling from the posterior distribution, an optimization process is instituted to construct bad outcomes. In one embodiment, an anytime algorithm is used to optimize and discover influential configurations. In another embodiment, backpropagation can be used to discover influential outcomes as further described below. Optimizing by discovering influential outcomes lessens the number of sampling required and increases the likelihood of discovering bad outcomes. The influential outcomes indicate how to perturb the state elements of the seed states. Once perturbed, the perturbed inputs are run through the simulator **120**. The perturbed inputs, like the seed states, are put into the outcome quantifier **128** to determine perturbed outcomes as the product of the perturbed probability and the perturbed cost. The perturbed cost is determined using the cost function and the perturbed probability is determined from the probability distributions **132**.

[0052] Further details for an example embodiment of the MPDM system are set forth. In this embodiment, non-holonomic motion models are used for each observed agent i as well as for the robot. The robot maintains a probabilistic estimate of each observed agents' state—i.e. its position, velocity, angular velocity and inferred policy. An agent's policy $\pi_{\text{sub},i}=(v_{\text{sub},\text{des}}, g_{\text{sub},\text{sub}})$, expresses an intent to move towards sub-goal $g_{\text{sub},\text{sub}}$ at a desired speed $v_{\text{sub},\text{des}}$. The collective state $x_{\text{sub},t} \in X$ consists of the states of the robot and all observed agents at time t . Throughout the disclosure, $x_{\text{sub},0}$ is referred to as the collective state of all agents and the robot's state at the current time. The probabilistic estimate $P(x_{\text{sub},0})$ is based on past observations of the pedestrians' positions. Several methods can be used for $P(x_{\text{sub},0})$ based on past trajectories of agents. In the example embodiment, a Kalman Filter is used to infer position and velocity; whereas, a Naive Bayes Classifier is used to infer an agent's policy. The robot's policy π is elected from amongst a set of closed-loop policies \mathcal{T} .

[0053] An initial sampled configuration $x_{\text{sub},0}$ is forward simulated H time-steps (through $t=1, \dots, H$), by recursively applying the transition function $T: X \rightarrow X$ to yield a trajectory

$$\begin{aligned} X(x_0) &= \{x_0, T(x_0), T^2(x_0), \dots, T^H(x_0)\} \\ &= \{x_0, x_1, x_2, \dots, x_H\} \end{aligned}$$

where $x_{\text{sub},t} \in X$ is the collective state consisting of the robot state plus all the agents at time t of the forward simulation. The transition function $T(\cdot)$ captures the trajectory that each agent is executing while at the same time considering the interactions with all other agents.

[0054] The cost function $C(X(x_{\text{sub},0}))$ assigns a scalar value to the outcome of a simulation. In the example embodiment, a cost function is used that penalizes the inconvenience the robot causes to other agents in the environment (Blame) along the predicted trajectory and rewards the robot's progress towards its goal (Progress). For blame, one can use the distance to the closest agent as a proxy for the potential disturbance caused to the environment by the robot.

$$[00012] B(X(\cdot, x_0)) = \max_{k=0}^s \sum_{j \neq r} u(\cdot \text{Math. } v_r \cdot \text{Math. } -) e^{-d_{r,j}(k)/\dots}$$

where $d_{r,j}(k)$ is the distance between the robot and agent j and $\|v_{r,j}(k)\|$ is the speed of the robot at time-step k . u is the step function with is 1 when the argument is ≥ 0 and 0—otherwise.

[0055] For progress, the robot is encouraged for the distance-made-good during the planning horizon.

$$[00013] PG(X(\cdot, x_0)) = (p_r(s) - p_r(0)) \cdot \text{Math. } e_{p_r \cdot \text{fwdarw. } g_r}$$

where $p_{r,j}(k)$ is the position of the robot at time-step k and $e_{p_{r,j} \cdot \text{fwdarw. } g_r}$ is the unit vector from the current position of the robot to the goal $g_{r,j}$. The resultant cost function is a linear combination of both

$$[00014] C(X(\cdot, x_0)) = - \alpha PG(X(\cdot, x_0)) + B(X(\cdot, x_0))$$

where α is a weighting factor. Other cost functions are also contemplated by this disclosure.

[0056] In risk-aware MPDM, the robot's policies are evaluated based on the most influential (likely and high-cost) outcome that may occur. Such outcomes are discovered by optimizing a probabilistic cost surface $\max \{P(x_{sub.0})C(X(\pi, x_{sub.0}))\}$, instead of the expected value of the cost function approximated by sampling.

[0057] Algorithm 1 below describes the policy election for risk-aware MPDM. Provided with a probability distribution over initial configurations, $P(x_{sub.0})$, a set of candidate policies, π , and a forward simulation budget, $N\pi$, each candidate policy is evaluated (scored) according to the most influential (worst case) outcome discovered within the computational budget.

[0058] The objective function $P(x_{sub.0})C(X)$ can have multiple local-minima depending on the number of agents and the complexity of the initial configuration. Finding the global maximum through exhaustive search is computationally infeasible due to the large state-space. The goal is to quickly find an influential configuration whose value is comparable to the global optimum even if it may not be the highest-valued configuration.

[0059] The following Algorithm 1 describes the policy election for risk-aware MPDM, that is, Algorithm 1 identifies influential or worse case outcomes.

TABLE-US-00001 Algorithm 1 Policy Election for Risk-aware MPDM

1: function Policy-	
Election Loop ($P(x)$, Π , $N_{sub.\pi}$)	2: for $\pi = \Pi$ do
3: while $n < N_{sub.\pi}$ do	3: Initialize $U_{sub.\pi}, n \leftarrow 0$
4: Sample $x_{sub.0} \sim P(x)$	5: U^* ,
6: $n_{sub.opt} \leftarrow \text{Optimize}(x_{sub.0}, \pi)$	6: U^* ,
7: $n \leftarrow n + n_{sub.opt}$	7: $U_{sub.\pi} \leftarrow$
8: $\max\{U^*, U_{sub.\pi}\}$	8: $U_{sub.\pi} \leftarrow$
9: end while	9: end while
10: end for	10: end for
11: $\pi^* \leftarrow \arg \min_{sub.\pi}$	11: $\pi^* \leftarrow \arg \min_{sub.\pi}$
12: end function	

[0060] The algorithm samples an initial configuration from $P(x_{sub.0})$ as indicate at line 5 and optimizes it, perturbing the sampled configuration iteratively towards increasingly influential outcomes until convergence to a local optima whose objective function value is U^* as indicated at line 6. The number of forward simulations $n_{sub.opt}$ used by an optimization procedure corresponds to its rate of convergence. Upon convergence, a new initial configuration is sampled and this process is repeated until the forward simulation budget $N_{sub.\pi}$, is consumed. The utility of a policy $U_{sub.\pi}$ is the most influential (highest-valued) configuration encountered. The policy with the least risk is elected.

[0061] With continued reference to FIG. 1, provided with a probability distribution over initial configurations $P(x_{sub.0})$, where $x_{sub.0}$ represents the state elements of all the objects at a current time, a set of candidate policies Π 136, and a forward simulations budget $N_{sub.\pi}$, each candidate policy 136 is evaluated and scored according to the most influential or worst case outcome discovered within the computational budget. The outcome (e.g., the product of the probability and cost) has multiple local minima depending on the number of objects and the complexity of the initial configuration. Finding the global maximum through exhaustive search is computationally infeasible due to the large state-space. Since the collisions are heavily penalized, a configuration

whose value is comparable to the global optimum is likely to be a collision and should therefore influence decision-making.

[0062] The simulator **120** is provided the seed states from the seed state generator **116** as further described below. Based on the seed states, the simulator **120** conducts forward simulations of a joint state until a time horizon. The joint state is the state values for all of the monitored objects and the controlled object **100**. The simulator **120** simulates the forward movement of each object in the environment for each of the potential policies **136** guiding the controlled object **100**. The forward simulation allows for the controlled object **100** to execute each of the possible policies **136** and iterate through all of the possible scenarios for each of the monitored objects that are perceivable by the controlled object **100**. That is, the simulator **120** forward simulates each of the possible variations of movements for each of the monitored objects while the controlled object **100** is executing different policies **136**. With each simulation, a cost and a probability may be determined for each of the policies **136**. The probability may be determined from the probability distributions **132** included in the data store **108**. The probability is based on the generated seed states.

[0063] The cost for each seed state and each policy **136** is calculated using a cost function. The cost function consists of two different components: (i) blame which captures the potential disturbance that the robot causes in the environment and (ii) progress which indicates progress made towards the goal of the controlled object **100**. The distance between the controlled object **100** and the closest monitored object is used as a proxy for the potential disturbance caused to the environment by the controlled object **100**. The controlled object **100** is also encouraged for the distance-made-good during a planning horizon. For a sampled initial configuration, the predicted trajectory is evaluated using a cost function. High value outcomes correspond to those where the controlled object **100** inconveniences monitored objects by driving too close to them, thus accumulating high blame. The controlled object **100** is also rewarded according to the progress it makes towards the goal.

[0064] The planning horizon is the amount of time into the future the simulator **120** predicts or simulates movement, e.g., three seconds. The cost function is calculated as a linear combination of both the blame and the progress. Additionally, to approximate the expected cost, a sampling technique is used. The sampling technique seeds the forward propagation of the joint state, which results in a set of different future trajectories. Therefore, the controlled object's **100** behavior reflects not only the mean state estimates of monitored objects, but also the uncertainty associated with the mean state estimates.

[0065] The outcome quantifier **128** calculates an outcome as a product of the probability and the cost for the generated seed state. The outcome indicates a score based on the policy **136** and the seed states. The score determines how good or bad choosing the respective policy **136** is under the seed state conditions. This forward simulation is conducted iteratively for each policy **136** and for many seed states to determine which policy **136** is most ideal under the presented conditions or configurations.

[0066] The perturbing module **124** also receives the seed states from the seed state generator **116** and perturbs these states. The perturbing module **124** optimizes the seed states by perturbing the seed states iteratively towards increasingly influential outcomes. An optimization procedure should converge in as few forward simulations (or iterations) as possible to its closest local optima.

[0067] Backpropagation may be used to perturb the seed states iteratively towards increasingly influential outcomes. Deep neural networks model complex functions by composing (chaining) relatively simple functions. Similarly, a forward simulation captures the complex dynamics of the system using simple one-step transition functions. Since the cost function is a linear combination of costs computed along the trajectory, the forward simulation can be conceptualized as a deep network that outputs a trajectory cost $C(X(x_{sub.0}))$ based on the input initial configuration. A function can accumulate the cost of a trajectory from the final time backwards to the initial time. This backpropagation can identify the most influential outcomes by starting at the final time and

finding the trajectory to work backwards to the initial configuration.

[0068] Let $L_{\text{sub.t}}(x_{\text{sub.t}})$ be the cost accrued at time-step t for the state $x_{\text{sub.t}}$. Define a function $\Phi(t, X)$ that accumulates the cost of a trajectory, from the final time H backwards to the initial time $t=0$ as

$$[00015] \quad \Phi(t, X) = \int_{T=t}^H L_T(x_T) dt. \quad (11)$$

The objective cost can be expressed as $J(X) = \Phi(0, X)$. One can formulate Φ recursively as:

$$[00016] \quad \Phi(t, X) = \Phi(t+1, X) + L_t(x_t). \quad (12)$$

One would want to compute $\nabla_{x_{\text{sub.0}}} J(X) = \nabla_{x_{\text{sub.0}}} \Phi(0, X)$. The gradient of the cost at time-step H is

$$[00017] \quad \nabla_{x_H} \Phi(H, X) = \frac{\partial \Phi(H, X)}{\partial x_H} = \frac{\partial L_H(x_H)}{\partial x_H} \quad (13)$$

One can compute the gradient iteratively from time-step H backwards to $t=0$ by applying (12) and expanding terms:

$$[00018] \quad \begin{aligned} \nabla_{x_t} \Phi(t, X) &= \frac{\partial \Phi(t, X)}{\partial x_t} = \frac{\partial \{ \Phi(t+1, X) + L_t(x_t) \}}{\partial x_t} \\ &= \frac{\partial \Phi(t+1, X)}{\partial x_t} + \frac{\partial L_t(x_t)}{\partial x_t} \\ &= \frac{\partial \Phi(t+1, X)}{\partial x_{t+1}} \frac{\partial x_{t+1}}{\partial x_t} + \frac{\partial L_t(x_t)}{\partial x_t} \\ &= \frac{\partial \Phi(t+1, X)}{\partial x_{t+1}} + \frac{\partial L_t(x_t)}{\partial x_t} \end{aligned} \quad (14)$$

Eqn. 14 can be used to efficiently compute $\nabla_{x_{\text{sub.0}}} J(X)$ as long as the gradient of transition function can be computed effectively.

[0069] This disclosure recognizes that the kinematic models used for the agents have an impact on the quality of the gradients. An alternative approach can use a simple double integrator model for all agents with heuristics to restrict lateral motion for more realistic simulation. While the simple model was useful for fast forward simulation, the heuristics contain hard thresholds that manifest as zeros in the matrix $\partial T(x_{\text{sub.t}})/\partial x_{\text{sub.t}}$. As a result, useful gradients are truncated (as highlighted by the box in Eqn. 14 hampering effective backpropagation. Thus, the use non-holonomic kinematic models that augment the agent's state with angular velocity to capture the effect of lateral forces is preferred. This model ensures the differentiability of T while maintaining realistic human motion in the forward simulation.

[0070] Specifically, the headed social force model (HSFM) is used for all the pedestrians and a unicycle-like model for the robot as described below. For the robot, the net force is computed using the SFM $f_{\text{sub.net.sup.r}}$, but due to the inherent constraints on a wheeled platform, transform $f_{\text{sub.net.sup.r}}$ into a compliant reference signal $(V_{\text{sub.ref}}, W_{\text{sub.ref}})$ for a lower-level velocity controller

$$[00019] \quad \begin{bmatrix} V_{\text{ref}} \\ W_{\text{ref}_{t+1}} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} f_{\text{net}}^r \quad (15)$$

The lookahead distance/determines the tendency of the robot to turn to compensate the lateral force. The robot's state is then propagated towards the reference signal using a first-order model for each of the independent wheel velocity controllers and a unicycle plant model.

[0071] Referring to FIG. 5, the proposed transition function layer $T(x_{\text{sub.t}})$ allows one to compute accurate gradients of the transition function. Eqn. 14 can now be implemented efficiently via backpropagation, where $\partial T(x_{\text{sub.t}})/\partial x_{\text{sub.t}}$ and $\partial L_{\text{sub.t}}(x_{\text{sub.t}})/\partial x_{\text{sub.t}}$ are computed during the forward propagation and cached.

[0072] FIGS. 6A and 6B illustrate one iteration of gradient descent using backpropagation through a simple initial configuration $x_{\text{sub.0}}$ consisting of two agents and the robot executing the Go-Solo policy towards its goal $g_{\text{sub.r}}$. The heuristic-based stochastic gradient method (SGA) computed

approximate gradients for each agent and perturbed one agent at a time to avoid divergence. In contrast, by computing accurate gradients, one can perturb all the agents simultaneously without divergence. The gradient also accounts for agent-agent interactions as well as static obstacles. [0073] Backpropagation, or any other method known in the art to identify influential outcomes, can be used to perturb the seed states and simulate using the perturbed seed states. The forward simulation of the perturbed seed state results in a perturbed outcome. The plurality of outcomes (product of probability and cost) for each of the possible policies **136** are used to identify which policy **136** has the best worst case scenario. That is, the best worst case scenario is, out of all the forward simulations for each of the policies **136**, the policy that has the best or most benign high-cost scenarios, including near misses. Once the best policy is identified, the controller **104** issues a command based on the selected policy.

[0074] There may be any number of policies **136** included in the data store **108**. In the example embodiment, the policies **136** are go solo, follow one of the monitored objects, and stop. The controlled object **100** executing the go solo policy treats all monitored objects as obstacles and uses a potential field based on a Social Force Model (SFM) to guide it towards the goal of the controlled object **100**. The SFM is described in “Social-aware robot navigation in urban environments,” in European Conference on Mobile Robotics, by G. Ferrer, A. Garrell, and A. Sanfeliu, 2013, pp. 331-336, which is incorporated by reference. In this embodiment, it is assumed that all monitored objects follow the go solo policy although policies of the monitored object may be inferred in other embodiments.

[0075] The controlled object **100** can also use the follow policy to deal with certain situations. In a crowd, the controlled object **100** may choose to follow one of the monitored objects sacrificing speed but delegating the task of finding a path to another object. Following could also be more suitable than overtaking the monitored object in a cluttered scenario as it allows the controlled object **100** to progress towards its goal without disturbing other agents, resulting in low blame.

[0076] According to the follow policy, the controlled object **100** chooses to follow one of the monitored objects, deemed a leader. The follow policy dictates that the controlled object **100** is attracted to the leader rather than the goal. The stop policy simply instructs the controlled object **100** to remain still. As the controlled object **100** is determining in real time which policy **136** results in the most benign high-cost scenarios, the policy **136** of the controlled object **100** is constantly changing and being reevaluated. Due to the low computational requirements of evaluating the set of policies **136**, the controlled object **100** can re-plan frequently. Frequent re-planning helps reduce the impact of uncertainty.

[0077] Referring now to FIG. 2, a diagram of an environment where a controlled object **100** would institute MPDM is illustrated. The environment includes the controlled object **100**, walls **204** or any kind of static object, a first monitored object **208**, a second monitored object **212**, and a target **216** or goal for the controlled object **100** to reach. As described with respect to FIG. 1, the controlled object **100** includes the controller **104** coupled to the data store **108**. In the environment, the controlled object **100** identifies the walls **204**, the first monitored object **208**, and the second monitored object **212** with the perception module **112**.

[0078] The perception module **112** determines values indicating position, velocity, and inferred goals for both of the monitored objects **208**, **212**. The determined values are states for the monitored objects **208**, **212**. Since the monitored objects **208**, **212** move freely through the environment, each monitored object's **208**, **212** goal cannot be perceived, only predicted. Additionally, each monitored object **208**, **212** has an unlimited number of trajectory options, some options are shown in FIG. 2. The first monitored object **208** has multiple options of trajectories **220**, **224** that the controlled object **100** must evaluate, and the second monitored object **212** has multiple trajectories **228**, **232** that the controlled object **100** must evaluate as well.

[0079] Based on the position, velocity, and inferred goal, the seed state generator **116** samples initial configurations or seed states of the monitored objects **208**, **212** to use during simulations of

each policy **136**. The simulator **120** determines a result or outcome of a scenario using the sampled initial configurations for each of the potential policies **136**. The perturbing module **124** then perturbs the initial configurations to create a worse outcome or a higher-cost event. The perturbing may be conducted using a number of known methods in the art, one of which is backpropagation, as discussed above. After perturbing the initial configurations, the simulator **120** conducts the simulation on the perturbed states, and then, with the outcome quantifier **128**, determines the outcome of the perturbed scenario. The initial configurations are perturbed and simulated for a predetermined period.

[0080] The predetermined period may be an amount of time that the controlled object **100** has to select a policy or a certain number of iterations before selecting a policy. After conducting simulations for the predetermined period, the controlled object **100** selects the policy whose population of discovered likely high-cost outcomes is more benign or desirable. That is, the selected policy includes the most benign worst case, high-cost events. Selecting one of the policies results in the controlled object **100** following one of multiple trajectories **236**, **240**. To advance toward the target **216**, the controlled object **100** may determine that going solo may be the policy with the most benign high-cost events, which may result in a direct trajectory toward the target **236**. Alternatively, the controlled object **100** may determine that following either the first monitored object **208** or the second monitored object **212** would result in the most benign high-cost events. Following the first monitored object **208** may result in a trajectory such as **240**, with the controlled object **100** accelerating to catch up behind the first monitored object **208** and remaining behind the first monitored object at a similar speed. As another option, the controlled object **100** may stop and stay in place until a different policy results in the most benign high-cost events.

[0081] Once the policy is selected, a command is issued to the controlled object **100**, such as change a trajectory to follow one of the one or more monitored objects, remain in the same position, move forward, decelerate; accelerate, or any other directive in accordance with the policy. Since the controlled object **100** is selecting policies in real time, the controlled object **100** is always computing which of the policies result in the most benign high-cost events and updating which policy to adhere to in accordance with the computations.

[0082] Referring now to FIG. **3**, a flowchart of an embodiment of the MPDM apparatus is illustrated. The controlled object **100** is always perceiving the states (e.g., locations, speed, etc.) of all the monitored objects in the environment and determining a trajectory based on the possible outcomes due to the initial configurations. To determine the trajectory, the controlled object **100** evaluates each of the policies **136**. First control chooses a policy to evaluate **300** from all of the potential policies **136**. Next, at step **304**, control receives state data for each of the monitored objects **208**, **212**. The state data is obtained by the perception module **112**. Seed states for each of the monitored objects **208**, **212** are generated at **308** using the seed state generator **116**. The seed states correspond to initial configurations that the simulations use as input. The simulator **120** then simulates using the chosen policy and the seed states at **312**. At step **316**, the outcome is quantified as the cost multiplied by the probability using the outcome quantifier **128**. As discussed above, the cost function is calculated as a combination of blame for disturbing objects in the environment as well as progress towards the target **216**. The probability is determined from the probability distributions **132**.

[0083] At step **320**, control determines whether a condition has been met. The condition may be a variety of predetermined conditions, such as an amount of time, a number of perturbations or iterations, a policy-specific condition depending on which policy is currently being implemented by the controlled object **100**, or any other condition that has been predetermined. In other words, step **320** controls the number of times the seed states are perturbed and simulated to determine which policy results in the most benign high-cost events.

[0084] If the condition has not been met, control continues to step **324** where the seed states are perturbed. That is, the elements of the seed states (i.e., position, speed, etc.) are perturbed

iteratively towards increasingly influential outcomes using a method such as backpropagation. This perturbation directs the perturbed seed state configurations towards outcomes that result in high-cost events. Then, at **328**, the simulator **120** simulates using the chosen policy and the perturbed seed states. The outcome quantifier **128** then quantifies a perturbed outcome as the product of the perturbed cost and the perturbed probability at **332**. The perturbed cost and perturbed probability are determined based on the perturbed seed states. The seed states are perturbed iteratively towards increasingly influential outcomes until the condition has been met. At that point, control determines whether each policy **136** has been simulated **336**. If not, control returns to the beginning to select a different policy at **200**.

[0085] Once control determines that each policy has been simulated at **336**, then scores are determined for each policy at **340**. The policy with the best score is selected at **344**. The best score can be described as the score indicating the fewest number of the most benign high-cost events. This ensures the best possible scenario for the controlled object **100**. That is, the controlled object **100** selects the policy whose population of discovered likely high-cost outcomes is more benign or desirable. For example, a more desirable policy includes a lower chance that the selected policy will result in a collision or a near miss, etc. Once the policy is selected, control issues a command associated with the policy to the controlled object **100** at **348**. As mentioned previously, the commands may be a command to accelerate, decelerate, etc. Control conducts the MPDM shown in the flowchart in real-time to determine, at all points in time, which policy **136** is best for the controlled object **100** to follow.

[0086] As a proof of concept, simulations were conducted. The simulated environment consists of an open space, freely traversed by 15 agents that can randomly change speed or direction while the robot tries to reach its goal. MPDM relies on quick decision making and replanning (every 300 nms) to react to sudden and unexpected changes in the environment.

[0087] A pedestrian can suddenly come to a stop, slow down or speed up. One can model this as a distribution over the preferred speed of each agent that is a mixture of two truncated Gaussians—one centered around the estimated most-likely current speed with a $\sigma=0.4$ m/s to account for speeding up or slowing down and a truncated half Gaussian with a peak at 0 and $\sigma=0.2$ m/s to account for coming to a sudden stop.

[0088] A pedestrian can also suddenly change direction without signaling. In order to account for uncertain direction for each agent, the robot assumes a Gaussian centered around the agent's estimated most-likely orientation and $\sigma=30^\circ$ that determines the agent's waypoint. All truncated Gaussians are restricted to $\mu \pm 1.5 \sigma$. A pedestrian's sub-goal is inferred from a set of salient points using a Naïve Bayes classifier.

[0089] For a sampled initial configuration, the predicted trajectory is evaluated using a cost function. High-cost outcomes correspond to those where the robot inconveniences other agents by driving too close to them, thus accumulating high Blame. The robot is also rewarded according to the Progress it makes towards the goal. Further description for these metrics can be found in “Fast discovery of influential outcomes for risk-aware MPDM” in Proceedings of the IEEE International Conference on Robotics and Automation, 2017 which is incorporated in its entirety by reference.

[0090] A dataset is generated consisting of 16,000 randomly chosen simulated scenarios where at least one agent was present within 5m of the robot. Then sort them based on the number of agents in the robot's neighbourhood. The objective function $P(x_{\text{sub.0}})C(X)$ is defined over innumerable possible initial configurations belonging to a high-dimensional continuous space that scales linearly with the number of agents considered. For each scenario, 2,000 random samples were optimized and the worst-case outcome was used to approximate the global optimum.

[0091] Next, vary the number of agents in the robot's vicinity, thus increasing the complexity of the scenario and the dimensionality of the state space. For reliable real-time policy evaluation, influential outcomes must be detected quickly. Estimate the number of iterations needed by each algorithm to achieve a certain fraction (50%) of the worst outcome in the dataset (find an

influential outcome). For each algorithm, the experiment is run 1,000 times on each scenario. Use bootstrap sampling (with replacement) on the data set to estimate the mean and standard error of their performance.

[0092] Stochastic Gradient Ascent computes approximate agent-specific gradients of a simplified cost function. In order to limit the divergence arising due to these approximations, the stochastic gradients are ranked using a heuristic function and only the most promising agent is perturbed at a time. Despite performing well in scenarios involving few agents, this method does not scale well to more challenging crowded settings. FIG. 7 shows that although all the algorithms take longer to find influential outcomes as the complexity of the environment grows, the performance of SGA deteriorates sharply for more than 3 agents. Beyond 6 agents, it performs as poorly as random sampling since it takes a long time to converge from a sampled initial configuration to a local optimum. Backpropagation, on the other hand, overcomes these limitations as it computes accurate gradients, and all agents can simultaneously be updated without divergence.

[0093] Through six hours of navigation in the simulated environment, it was demonstrated that the proposed approach, unlike SGA, can reliably evaluate a large policy set. Each simulation ‘epoch’ consists of a random initialization of agent states followed by a 5 minute simulated run at a granularity $\Delta t=0.15s$. In the simulator, the observations z are modeled using a stationary Gaussian distribution with uncorrelated variables for position, speed and orientation for the agent.

Parameterize this uncertainty by a scale factor $\{\sigma_{p.x}, \sigma_{p.y}, \sigma_{|v|}, \sigma_{\theta}\} = \{10 \text{ cm}, 10 \text{ cm}, 10 \text{ cm/s}, 15^\circ\}$. The corresponding diagonal covariance matrix is denoted by $\text{diag}(\sigma_{p.x}, \sigma_{p.y}, \sigma_{|v|}, \sigma_{\theta})$. Do not perturb the goal and assume no angular velocity (ignoring any uncertainty). These uncertainties are propagated in the posterior state estimation $P(x|z)$.

[0094] The simulation experiments are run on an Intel i7 processor and 8 GB RAM to mimic the computational capabilities of the robot. In order to react to sudden changes, MPDM relies on quick re-planning. The robot must replan every 300 ms for effective real-time navigation. The performance of risk-aware MPDM is evaluated using 2 candidate sets of policies—a large candidate set with 10 policies, and a small set with 2 policies: [0095] 1) 2 Policies—{Go-Solo, Stop}—The robot evaluates going straight towards the goal at maximum speed (1.5 m/s) and stops if it senses danger. [0096] 2) 10 Policies—{(Fast, Medium, Slow) × (Straight, Left, Right), Stop}—Rather than going straight towards the goal at maximum speed, the MPDM may also choose to go at Medium speed (0.9 m/s) or Slowly (0.2 m/s). Simultaneously, the robot can also choose to create a sub-goal to the Left or Right of the goal instead of going Straight to the goal as in Go-Solo.

[0097] Record the Time Stopped per goal reached, as well as the Blame normalized by the distance to goal (Blame per meter traveled). Time Stopped indicates the failure of the planner to find a safe policy. With a larger policy set, the robot is more likely to find a safe policy, and Stops less often. However, if the robot cannot evaluate its policy set quickly enough, it is unable to react to sudden changes in the environment and accumulates Blame. Ideally, one would like a robot navigate safely (low Blame), with minimal Stop-and-Go motion.

[0098] FIG. 8 shows how the inefficiencies in SGA become a performance bottleneck. While SGA can navigate safely (low Blame) with the small policy set, it often fails to find safe policies and stops. With 10 policies, SGA fails to find influential outcomes fast enough resulting in high Blame. The proposed method, BP can reliably evaluate the large policy set in real-time, which significantly improves navigation performance.

[0099] In the example embodiment, the system is implemented on the MAGIC robot, a differential drive platform equipped with a Velodyne VLP-16 laser scanner used for tracking and localization. LCM is used for inter-process communication. Every 300 ms, MPDM evaluates a set of policies and chooses the least risky one. Although the policy election is slow, the robot is responsive as the policies themselves run at 50 Hz.

[0100] Seven volunteers were asked to move towards marked points around an open space for 45

minutes. FIG. 9 shows data from 90 minutes of real-world experiments in which volunteers were asked to repeat three fixed scenarios while the robot made its way towards its goal. For both, our proposed approach as well as SGA, each scenario was repeated for 15 minutes. As observed in simulation, SGA was too slow to evaluate the larger policy set reliably and was unsafe to deploy on our robot. Using SGA with two policies (purple), the robot fails to find safe policies and stops often. Our proposed method (green) can reliably evaluate **10** policies in real-time (similar Blame as compared to SGA with just two policies) and as a result, it is more likely to find safe policies (low Time Stopped).

[0101] The foregoing description is merely illustrative in nature and is in no way intended to limit the disclosure, its application, or uses. The broad teachings of the disclosure can be implemented in a variety of forms. Therefore, while this disclosure includes particular examples, the true scope of the disclosure should not be so limited since other modifications will become apparent upon a study of the drawings, the specification, and the following claims. It should be understood that one or more steps within a method may be executed in different order (or concurrently) without altering the principles of the present disclosure. Further, although each of the embodiments is described above as having certain features, any one or more of those features described with respect to any embodiment of the disclosure can be implemented in and/or combined with features of any of the other embodiments, even if that combination is not explicitly described. In other words, the described embodiments are not mutually exclusive, and permutations of one or more embodiments with one another remain within the scope of this disclosure.

[0102] Spatial and functional relationships between elements (for example, between modules, circuit elements, semiconductor layers, etc.) are described using various terms, including “connected,” “engaged,” “coupled,” “adjacent,” “next to,” “on top of,” “above,” “below,” and “disposed.” Unless explicitly described as being “direct,” when a relationship between first and second elements is described in the above disclosure, that relationship can be a direct relationship where no other intervening elements are present between the first and second elements, but can also be an indirect relationship where one or more intervening elements are present (either spatially or functionally) between the first and second elements. As used herein, the phrase at least one of A, B, and C should be construed to mean a logical (A OR B OR C), using a non-exclusive logical OR, and should not be construed to mean “at least one of A, at least one of B, and at least one of C.”

[0103] In the figures, the direction of an arrow, as indicated by the arrowhead, generally demonstrates the flow of information (such as data or instructions) that is of interest to the illustration. For example, when element A and element B exchange a variety of information but information transmitted from element A to element B is relevant to the illustration, the arrow may point from element A to element B. This unidirectional arrow does not imply that no other information is transmitted from element B to element A. Further, for information sent from element A to element B, element B may send requests for, or receipt acknowledgements of, the information to element A.

[0104] In this application, including the definitions below, the term “module” or the term “controller” may be replaced with the term “circuit.” The term “module” may refer to, be part of, or include: an Application Specific Integrated Circuit (ASIC); a digital, analog, or mixed analog/digital discrete circuit; a digital, analog, or mixed analog/digital integrated circuit; a combinational logic circuit; a field programmable gate array (FPGA); a processor circuit (shared, dedicated, or group) that executes code; a memory circuit (shared, dedicated, or group) that stores code executed by the processor circuit; other suitable hardware components that provide the described functionality; or a combination of some or all of the above, such as in a system-on-chip.

[0105] The module may include one or more interface circuits. In some examples, the interface circuits may include wired or wireless interfaces that are connected to a local area network (LAN), the Internet, a wide area network (WAN), or combinations thereof. The functionality of any given module of the present disclosure may be distributed among multiple modules that are connected via

interface circuits. For example, multiple modules may allow load balancing. In a further example, a server (also known as remote, or cloud) module may accomplish some functionality on behalf of a client module.

[0106] Some or all hardware features of a module may be defined using a language for hardware description, such as IEEE Standard 1364-2005 (commonly called “Verilog”) and IEEE Standard 1076-2008 (commonly called “VHDL”). The hardware description language may be used to manufacture and/or program a hardware circuit. In some implementations, some or all features of a module may be defined by a language, such as IEEE 1666-2005 (commonly called “SystemC”), that encompasses both code, as described below, and hardware description.

[0107] The term code, as used above, may include software, firmware, and/or microcode, and may refer to programs, routines, functions, classes, data structures, and/or objects. The term shared processor circuit encompasses a single processor circuit that executes some or all code from multiple modules. The term group processor circuit encompasses a processor circuit that, in combination with additional processor circuits, executes some or all code from one or more modules. References to multiple processor circuits encompass multiple processor circuits on discrete dies, multiple processor circuits on a single die, multiple cores of a single processor circuit, multiple threads of a single processor circuit, or a combination of the above. The term shared memory circuit encompasses a single memory circuit that stores some or all code from multiple modules. The term group memory circuit encompasses a memory circuit that, in combination with additional memories, stores some or all code from one or more modules.

[0108] The term memory circuit is a subset of the term computer-readable medium. The term computer-readable medium, as used herein, does not encompass transitory electrical or electromagnetic signals propagating through a medium (such as on a carrier wave); the term computer-readable medium may therefore be considered tangible and non-transitory. Non-limiting examples of a non-transitory computer-readable medium are nonvolatile memory circuits (such as a flash memory circuit, an erasable programmable read-only memory circuit, or a mask read-only memory circuit), volatile memory circuits (such as a static random access memory circuit or a dynamic random access memory circuit), magnetic storage media (such as an analog or digital magnetic tape or a hard disk drive), and optical storage media (such as a CD, a DVD, or a Blu-ray Disc).

[0109] The apparatuses and methods described in this application may be partially or fully implemented by a special purpose computer created by configuring a general purpose computer to execute one or more particular functions embodied in computer programs. The functional blocks and flowchart elements described above serve as software specifications, which can be translated into the computer programs by the routine work of a skilled technician or programmer.

[0110] The computer programs include processor-executable instructions that are stored on at least one non-transitory computer-readable medium. The computer programs may also include or rely on stored data. The computer programs may encompass a basic input/output system (BIOS) that interacts with hardware of the special purpose computer, device drivers that interact with particular devices of the special purpose computer, one or more operating systems, user applications, background services, background applications, etc.

Claims

1. A computer-implemented method for issuing a command to a controlled object in an environment, comprising: for each policy from a set of policies, wherein the policy specifies the command for the controlled object to implement and the command directly affects motion of the controlled object: receiving a state estimate for each of one or more monitored objects and the controlled object, wherein each state estimate includes state elements, and wherein the state elements are indicative of a position of the respective object and a velocity of the respective object;

generating seed states for each of the one or more monitored objects and the controlled object; simulating movement of the one or more monitored objects and the controlled object using the seed states, wherein the simulation includes the controlled object executing the respective policy; assigning a cost to outcome of the simulation with the seed states; determining a probability associated with the seed states; quantifying an outcome of the simulation with the seed states based on the cost and the probability; simulating movement of the one or more monitored objects and the controlled object using perturbed inputs by: perturbing at least one state element of the seed states to determine perturbed seed states; simulating movement of the one or more monitored objects and the controlled object using the perturbed seed states; assigning a perturbed cost to outcome of the simulation with the perturbed seed states; determining a perturbed probability associated with the perturbed seed states; quantifying a perturbed outcome of the simulation with the perturbed seed states based on the perturbed cost and the perturbed probability; and repeating the simulating movement of the one or more monitored objects and the controlled objects with perturbed inputs until a predetermined condition is met, thereby generating a plurality of perturbed outcomes; determining a policy score for the respective policy, wherein the policy score correlates to the perturbed outcome having highest value amongst the plurality of perturbed outcomes for the respective policy; selecting a given policy from the set of policies, wherein the given policy has most benign outcome amongst the policies in the set of policies; and issuing the command to the controlled object in accordance with the given policy.

2. The method of claim 1 wherein the perturbing at least one state element of the seed states maximizes a product of the perturbed cost and the perturbed probability.

3. The method of claim 1 further comprises perturbing at least one state element using backpropagation.

4. The method of claim 1 wherein simulating movement of the one or more monitored objects further comprises representing trajectory of an object using a differentiable function.

5. The method of claim 4 further comprises representing trajectory of an object by recursively applying a transition function over a series of time steps, where the transition function is defined such that objects are repelled by other agents and attracted towards a goal in accordance with a social force model.

6. The method claim 4 wherein simulating movement of the one or more monitored objects includes determining perturbed seed states by iteratively computing gradient for each time step in the series of time steps with respect to the perturbed seed states.

7. The method of claim 5 wherein the given policy avoids a set of undesired outcomes, and wherein the set of undesired outcomes includes at least one of (i) a collision between the controlled object and one of the one or more monitored objects; and (ii) the controlled object being within a predetermined distance of the one or more monitored objects.

8. The method of claim 1 wherein the state elements are represented by a probability distribution.

9. The method of claim 8 wherein one of the state elements is indicative of a goal of the controlled object.

10. The method of claim 9 wherein the cost is determined using a blame metric, wherein the blame metric is a function of a distance between the controlled object and one of the one or more monitored objects and the velocity of the controlled object.

11. The method of claim 10 wherein the cost is determined using a progress toward the goal of the controlled object.

12. The method of claim 1 wherein the received state estimate for each of the one or more monitored objects is a probabilistic estimate of the one or more monitored objects based on a perception module.

13. The method of claim 1 wherein the set of policies include at least one of the following commands for the controlled object to: (i) change a trajectory to follow one of the one or more monitored objects; (ii) remain in the same position; (iii) move forward; (iv) decelerate; and (v)

accelerate.

14. A controlled object in an environment comprising: a controller configured to, for each policy from a set of policies, determine a best policy, wherein the policy specifies a command for the controlled object to implement, and wherein the command indicates a direction of the controlled object; a non-transitory computer-readable medium that stores instructions that, when executed by a processor, cause the processor to: receive, from a perception module, a state estimate for each of one or more monitored objects and the controlled object, wherein each state estimate includes state elements, and wherein the state elements are indicative of a position of the respective object and a velocity of the respective object; generate, via a seed state generator, seed states for each of the one or more monitored objects and the controlled object; simulate, via a simulator, movement of the one or more monitored objects and the controlled object using the seed states, wherein the simulation includes the controlled object executing the respective policy; quantify, via an outcome quantifier, an outcome for the seed states by assigning a cost to the outcome and determining a probability associated with the seed states; simulate, via the simulator, movement of the one or more monitored objects and the controlled objects using perturbed inputs by: perturbing, via a perturbing module, at least one state element of the seed states using backpropagation to determine perturbed seed states, wherein the perturbing maximizes the product of a perturbed cost and a perturbed probability; simulating, via the simulator, movement of the one or more monitored objects and the controlled object using the perturbed seed states to calculate the perturbed cost associated with the perturbed seed states; quantifying, via the outcome quantifier, a perturbed outcome for the perturbed seed states as a perturbed product of the perturbed cost and the perturbed probability associated with the perturbed seed states; and repeating, via the controller, the simulating movement of the one or more monitored objects and the controlled objects with perturbed inputs until a predetermined condition is met, thereby generating a plurality of perturbed products; determine, via the controller, a policy score for the respective policy, wherein the policy score correlates to the perturbed outcome having highest value amongst the policies in the set of policies; select, via the controller, a given policy from the set of policies, wherein the given policy has most benign outcome amongst the policies in the set of policies; and issue, via the controller, the command to the controlled object in accordance with the given policy.

15. The controlled object of claim 14 further comprise simulating movement of the one or more monitored objects by representing trajectory of an object using a differentiable function.

16. The controlled object of claim 15 further comprises representing trajectory of an object by recursively applying a transition function over a series of time steps, where the transition function is defined such that objects are repelled by other agents and attracted towards a goal in accordance with a social force model.

17. The controlled object of claim 15 wherein simulating movement of the one or more monitored objects includes determining perturbed seed states by iteratively computing gradient for each time step in the series of time steps with respect to the perturbed seed states.

18. The controlled object of claim 14 wherein the set of policies include at least one of the following commands for the controlled object to: (i) change lanes, (ii) proceed through an intersection, (iii) merge into traffic at a position related to one or more vehicles within the other flow of traffic.
