

Autonomous navigation of stratospheric balloons using reinforcement learning

<https://doi.org/10.1038/s41586-020-2939-8>

Received: 1 April 2020

Accepted: 29 September 2020

Published online: 2 December 2020

 Check for updates

Marc G. Bellemare^{1✉}, Salvatore Candido^{3✉}, Pablo Samuel Castro¹, Jun Gong³, Marlos C. Machado¹, Subhodeep Moitra¹, Sameera S. Ponda³ & Ziyu Wang²

Efficiently navigating a superpressure balloon in the stratosphere¹ requires the integration of a multitude of cues, such as wind speed and solar elevation, and the process is complicated by forecast errors and sparse wind measurements. Coupled with the need to make decisions in real time, these factors rule out the use of conventional control techniques^{2,3}. Here we describe the use of reinforcement learning^{4,5} to create a high-performing flight controller. Our algorithm uses data augmentation^{6,7} and a self-correcting design to overcome the key technical challenge of reinforcement learning from imperfect data, which has proved to be a major obstacle to its application to physical systems⁸. We deployed our controller to station Loon superpressure balloons at multiple locations across the globe, including a 39-day controlled experiment over the Pacific Ocean. Analyses show that the controller outperforms Loon's previous algorithm and is robust to the natural diversity in stratospheric winds. These results demonstrate that reinforcement learning is an effective solution to real-world autonomous control problems in which neither conventional methods nor human intervention suffice, offering clues about what may be needed to create artificially intelligent agents that continuously interact with real, dynamic environments.

Superpressure balloons¹ can autonomously operate in the stratosphere for months, making them a cost-effective platform for communication, Earth observation, gathering meteorological data and other applications. The altitude of a superpressure balloon is determined by its density relative to the ambient atmosphere. In a Loon superpressure balloon, vertical motion is achieved by pumping air ballast in and out of a fixed-volume envelope, and horizontal motion is dictated by the winds at the balloon's location. To navigate, a flight controller must therefore ascend and descend to find and follow favourable wind currents (Fig. 1a).

Despite these simple dynamics, long-term balloon control is challenging. The task of 'station-keeping', for example, involves maintaining the balloon above a fixed ground location. To succeed, an indirect flight path must be taken through the wind field (Fig. 1b). Station-keeping also requires managing power over the day–night cycle, as descending uses solar energy stored in on-board batteries. Sparse wind measurements result in a phenomenon called partial observability⁹, and a good controller must weigh the costs and benefits of gathering distal observations. The nature of this partial observability alone limits the usefulness of conventional control techniques².

We used reinforcement learning⁵ to train a flight controller from simulations. Reinforcement learning excels at producing control strategies that can handle high-dimensional, heterogeneous inputs and optimize long-term objectives—for example, resulting in superhuman game play^{10–12}. Reinforcement learning has proven effective in optimization problems in which a model of the process is available^{13–16}, and has

shown promising results in simulations of real applications and back-testing^{17–20} with some commercial success in internet applications^{21,22}.

However, learning from imperfect data results in behavioural inaccuracies that compound over the planning horizon^{23,24}. This poses an important difficulty in scaling up to autonomous physical platforms such as ours, where acquiring flight data is slow and expensive, and where decisions have consequences more than 24 h into the future. Past results on autonomous flight, in comparison, have optimized shorter-horizon objectives such as trajectory following^{25,26} or maximizing upwards velocity²⁷. In the extreme, a balloon may need weeks of roundabout flight to recover from control errors, without the benefit of a reset mechanism such as is common in robotics experiments^{7,28–31}. Finally, the controller must cope with ageing-related changes such as helium loss and battery fatigue.

Station-keeping

We say that a balloon is successfully station-keeping when within 50 kilometres of its station, a distance at which it can comfortably communicate with a ground device. The availability of diverse opposing winds at different altitudes facilitates station-keeping. In the tropics, this diversity varies seasonally and is affected by large-scale weather phenomena^{32–34}. However, short-term forecasts for the region can substantially differ from actual measurements; wind-heading prediction errors greater than 90° are frequent, and are greatest at the equator^{35,36}. This makes exploration integral to an effective control strategy.

¹Brain Team, Google Research, Montreal, Quebec, Canada. ²Brain Team, Google Research, Toronto, Ontario, Canada. ³Loon, Mountain View, CA, USA. ✉e-mail: bellemare@google.com; scandido@loon.com

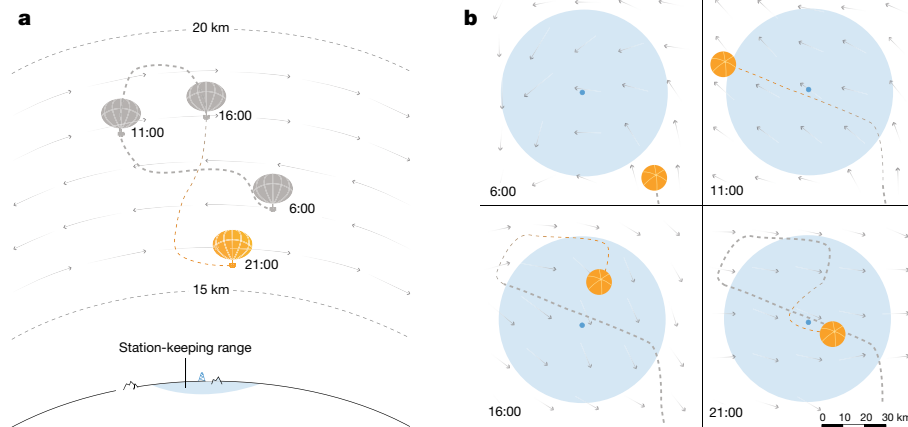


Fig. 1 | Station-keeping with a superpressure balloon. **a**, Schematic of a superpressure balloon navigating a wind field. The balloon remains close to its station by moving between winds at different altitudes. Its altitude range is indicated by the upper and lower dashed lines. **b**, The balloon's flight path,

viewed from above. The station and its 50-km range are shown in light blue. Shaded arrows represent the wind field. The wind field constantly evolves, requiring the balloon to replan at regular intervals.

Loon's previous hand-crafted algorithm, colloquially named StationSeeker, provides insight into what makes a good controller. StationSeeker tracks winds with headings that form an acute angle with the direction to the station, effectively tacking towards its destination. Once the balloon is in range, it seeks slow-moving winds. Decisions are made by maximizing a score function that incentivizes exploration (see Methods). Although heuristic, this controller has been tuned extensively and has reliably navigated Loon balloons for over 150,000 flight hours.

A better controller spends a greater fraction of time successfully station-keeping. To reserve energy for the balloon's payload, it should also minimize power consumption. We therefore require that deployed controllers use on average no more power than StationSeeker, which is calibrated for normal operations.

We encode this objective using a reward, r , that is maximal ($r=1$) when the balloon is within $\rho=50$ km of its station. This reward is associated with the balloon state s_t at time t and is provided in response to an action a_t (ascend, descend or stay). Given a discount factor $\gamma < 1$, an optimal controller maximizes the expected discounted sum of future rewards, or 'return'

$$R_s = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s \right],$$

where \mathbb{E} indicates the expected value. R_s characterizes the long-term value of a flight controller from an initial state s onwards.

Although setting the reward to $r=0$ outside the 50 km range describes the objective, softening the transition at the boundary results in improved controllers. We reduce the reward r to a constant $c_{\text{cliff}} < 1$ at 50 km and decay it by a half every $\tau=100$ km (see Methods). To incentivize power efficiency, r is further decreased when consuming energy. An ablation study confirms that the learning is robust to a wide range of reward parameters (Fig. 2a).

Wind measurements are typically only available at the balloon's current position and in its wake, where they have been obtained from instruments. To extrapolate to other positions, we use a Gaussian process³⁷ to blend the balloon's measurements with forecasts from the European Centre for Medium-Range Weather Forecast (ECMWF), using the wind forecast as a prior mean. The variance of the posterior distribution quantifies the uncertainty of different wind estimates. As inputs to our controller, we encode the magnitude and relative bearing of winds directly above and below the balloon, at 181 pressure levels ranging from 5 kPa to 14 kPa (15–20 km equivalent altitude). This wind column also forms the bulk of StationSeeker's inputs. Using wind estimates outside of this column increases the computational costs

without substantially affecting performance, because the wind field varies slowly with latitude and longitude and also because the Gaussian process integrates measurements from nearby balloons. We encode the uncertainty at each pressure level to provide an approximate belief state³⁸. By treating uncertainty as an input, we avoid explicitly enumerating plausible scenarios—a considerable computational advantage over search methods. Finally, we encode 16 ambient variables important to efficient flight (Extended Data Table 1).

The wind column encoding is centred at the balloon's altitude, so that its frame of reference is independent of the balloon's absolute coordinates. This provides the learning system with a useful inductive bias that takes advantage of natural symmetries and supports a simple strategy: ascend or descend when the winds above or below offer better returns, stay if they do not. Because poor decisions take the balloon to a lower-return state, this strategy corrects for many mistakes by design. We thus expect it to perform well in different wind conditions and handle discrepancies between real and simulated environments.

A large supply of realistic training data is key to successful reinforcement learning. Data from previous balloon flights are inadequate as they cannot be used to evaluate large deviations from historical behaviour. On the other hand, generating sufficiently accurate data from a physical atmospheric simulation is computationally prohibitive. Instead, we create plausible wind data based on the ECMWF's ERA5 global reanalysis dataset³⁹, which reinterprets historical weather observations using numerical models. ERA5 provides baseline winds that are modified with procedural noise⁴⁰ to generate high-resolution wind fields. By varying the random seed that drives the procedural noise, we can generate an arbitrary number of scenarios and emulate forecasting errors. The end product bears resemblance to data augmentation^{6,7}, which is known to improve the robustness of reinforcement-learning controllers to modelling discrepancies.

We model the dynamics of a Loon superpressure balloon within these plausible wind fields to obtain a simulator for training and evaluating flight controllers. A trial consists of two simulated days of station-keeping at a fixed location, during which controllers receive inputs and emit commands at 3-min intervals. Flight controllers are thus exposed to diurnal cycles and scenarios in which the balloon must recover from difficult overnight conditions. These realistic flight paths come at the cost of relatively slow simulation—roughly 40 Hz on data-centre hardware. In comparison, the Arcade Learning Environment (ALE) benchmark⁴¹ operates at over 8,000 Hz.

Our controller uses action values to predict the rewards resulting from its decisions. The action value $R_{s,a}$ estimates the expected return

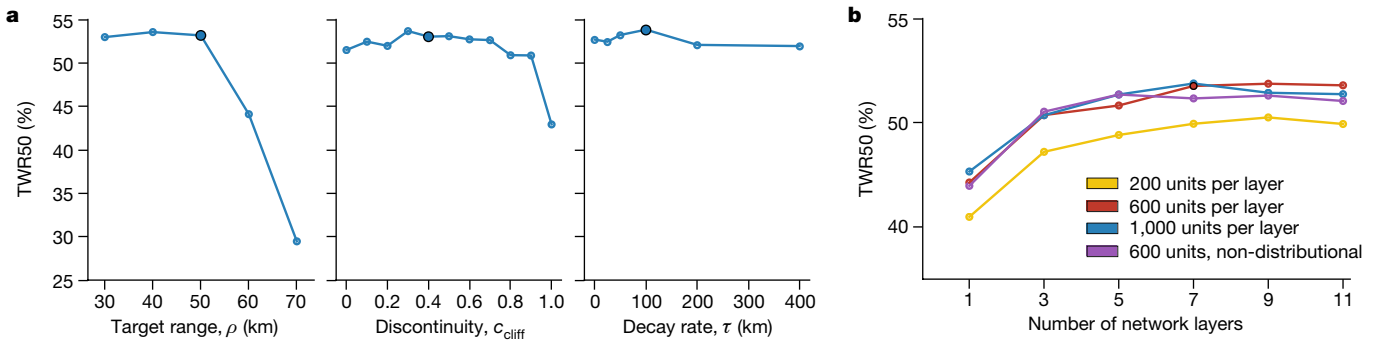


Fig. 2 | Effect of parameters on controller performance. **a**, Performance of reinforcement-learning controllers trained with different reward parameters. Performance is measured in terms of the fraction of time spent within 50 km of the station (TWR50), evaluated using our simulation benchmark (averaged

over 6,000 simulations; see text). Each parameter is varied in isolation; the deployed controller's parameter settings are indicated by a black outline. **b**, Performance as a function of network size.

obtained when the action a is selected from a state s . During operation, the controller behaves 'greedily', selecting the action that has the highest estimate. Action-value estimation is particularly well suited to the problem of station-keeping, in which there are few actions to evaluate. Unlike direct policy optimization schemes^{42,43}, it is also trivially combined with experience replay⁴⁴ to reuse simulator interactions and achieves greater data efficiency.

To obtain a state-of-the-art controller, we leveraged recent developments from the field of deep reinforcement learning, which emphasizes the use of deep neural networks in the learning process. Our controller estimates its action values using a feed-forward network with seven layers of 600 rectified linear units⁴⁵ each; the weights of the network are learned using the distributional QR-DQN algorithm⁴⁶. Using smaller networks or a non-distributional algorithm was found to degrade performance (Fig. 2b).

We train the neural network in a distributed setting⁴⁷. Data are generated from 100 parallel simulations and stored in an array of circular replay buffers from which minibatches are sampled and provided to a learner process⁴¹. Distributed training overcomes the low simulation rate and enables training within a reasonable amount of time. The training process alternates between its greedy policy and a momentum-based exploratory policy (see Methods).

Evaluation

We evaluate controllers in simulation using a 6,000-flight benchmark. This benchmark serves as a reliable proxy for actual flight performance, and allows fast iterations over the controller design. By varying hyper-parameters, we obtain a number of controllers that exhibit different

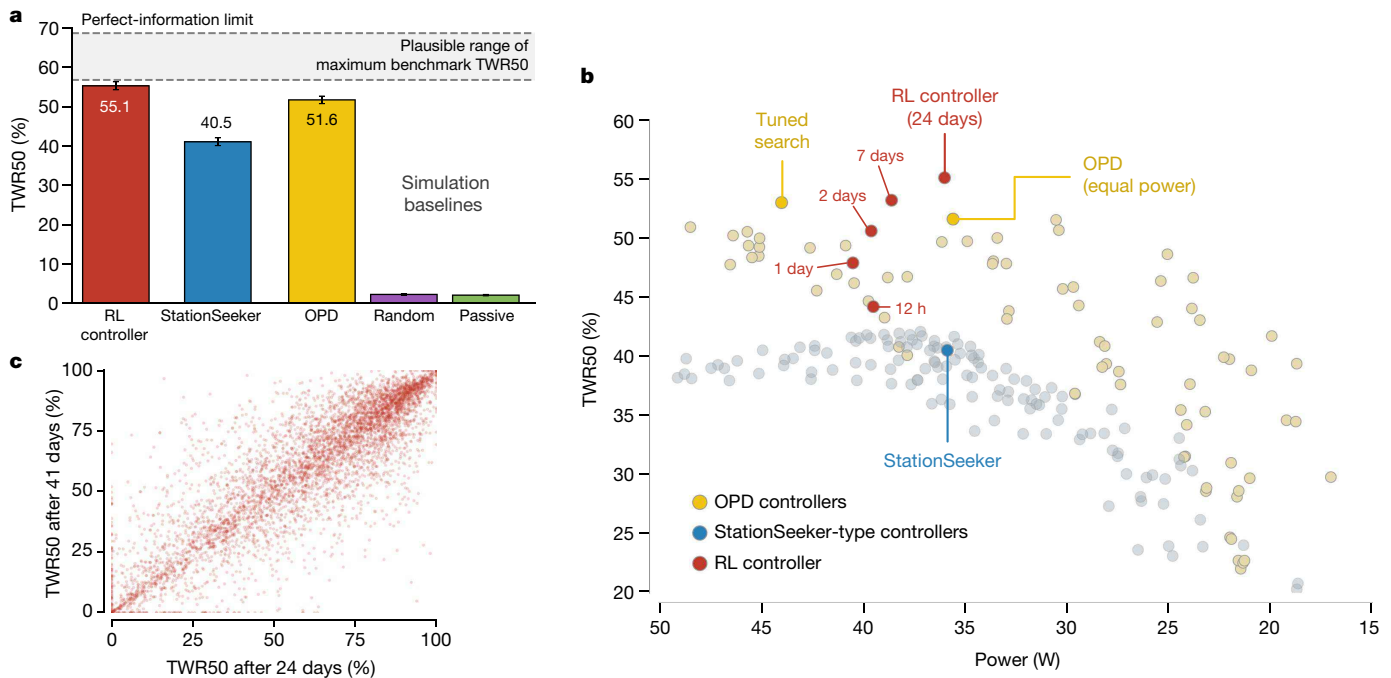


Fig. 3 | Performance profile of the reinforcement-learning controller in simulation. **a**, Proportion of flight time spent within 50 km of the station (TWR50) by different controllers in the simulation benchmark (averaged over 6,000 simulations); a higher TWR50 reflects better performance. Also included are controllers that select actions uniformly at random and passively drift. Error bars show 99.9% confidence intervals. The shaded area indicates the estimated range for the maximum TWR50 in the benchmark. RL, reinforcement learning. **b**, TWR50 and power consumption for alternative

parametrizations of StationSeeker and OPD. The controllers from **a**, and the reinforcement-learning controller at different points in training, are indicated. **c**, Per-flight performance of the reinforcement-learning controller after 24 and 41 days of training. Points indicate the two controllers' TWR50 for specific flights from the benchmark. Both controllers achieve 55.1% TWR50, but differ substantially on a per-flight basis, illustrating the partially observable nature of station-keeping.

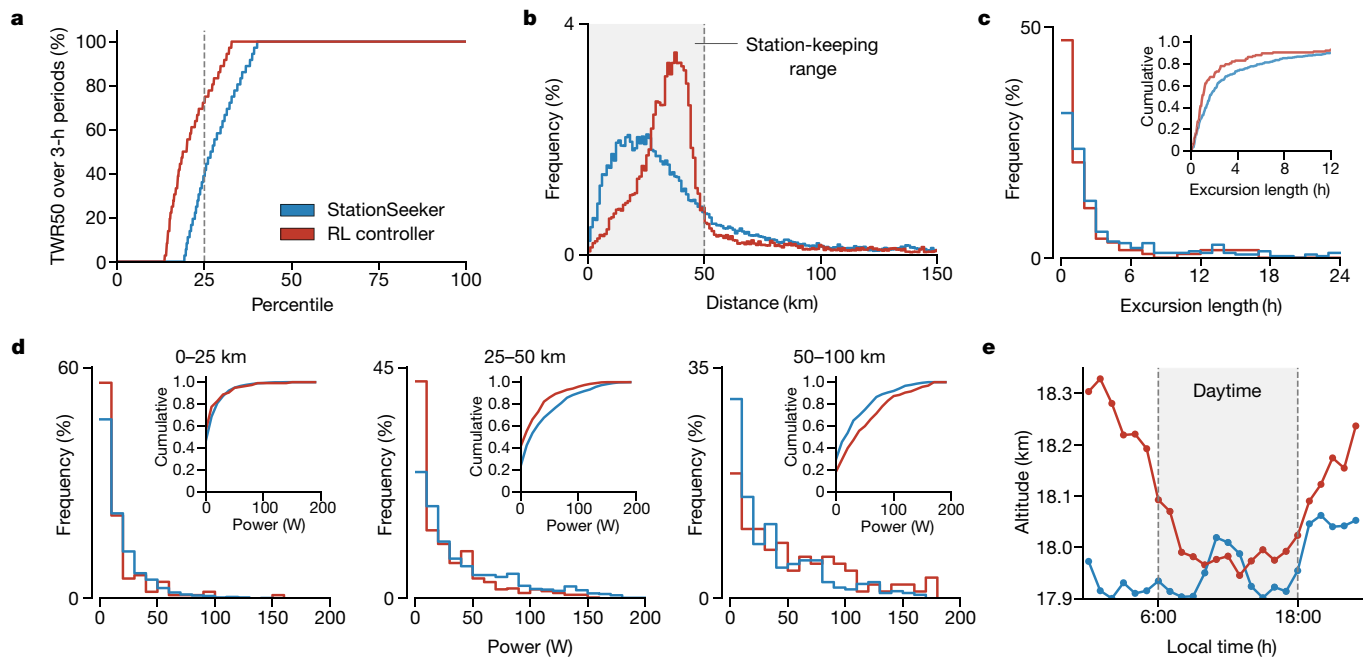


Fig. 4 | Flight characteristics of controllers tested in the Pacific Ocean experiment. **a**, Distribution of time spent within 50 km (TWR50) across 3-h periods. The panel shows the cumulative fraction of periods (x axis) during which up to a given fraction of time was spent within 50 km of the station (y axis). A left-shifted curve indicates higher performance and the reinforcement-learning controller has a lower 25th-percentile TWR50. **b**, Histogram of distances to the station across 3-h periods (bin width: 1 km). **c**, Histogram of duration of excursions (bin width: 1 h). An excursion begins when the balloon leaves the 50-km range and ends when it returns within range.

The distribution of StationSeeker’s excursion lengths shows a heavier tail. Inset shows cumulative frequency to 12 h (89.7% and 92.5% of data for the reinforcement-learning controller and StationSeeker, respectively). **d**, Histogram of power usage across 3-h periods (bin width: 10 W) within the inner and outer rings (0–25 km and 25–50 km) and outside the range (50–100 km). Insets show cumulative frequencies, illustrating how the controllers use energy for different purposes. **e**, Average altitude over the day (1-h increments). The shaded area indicates the period during which solar energy is available. Local time is –8 UTC.

trade-offs between station-keeping performance and energy usage. The best controller satisfying our power requirement achieves a score of 55.1% TWR50 (time within radius 50 km) on the benchmark. By comparison, StationSeeker achieves 40.5% TWR50 (Fig. 3a). A 1% gain corresponds to 14.4 additional minutes of station-keeping in a 24-h period, and so the difference amounts to a substantial 3.5 h per day average improvement in time spent near the station.

We estimate that the maximum achievable TWR50 on this benchmark is somewhere between 56.8% and 68.7% (see Methods). The upper limit corresponds to the performance of a tree-search controller with perfect wind information, and reflects the limitations of wind-based navigation. Even if favourable winds exist, partial observability implies that an agent that behaves optimally with respect to its knowledge must still sometimes gamble. This is reflected in the network’s distributional prediction, which we use to obtain the 56.8% lower limit. The per-flight TWR50 of equally performing controllers can vary substantially (Fig. 3c), which suggests that the true maximum lies closer to the lower limit, implying that our controller is close to optimal in simulation.

As a strong competitor, we applied a tree-search algorithm (optimistic deterministic planning, OPD⁴⁸) to a coarse approximation of flight dynamics, used for computational reasons (see Methods for modelling details). At equal power consumption, the reinforcement-learning controller outperforms a controller based on OPD (51.6% TWR50; Fig. 3b). Even a tuned-search controller that uses all available power remains worse (53.0%, Mann–Whitney U test $n = 6,000$, $P < 10^{-4}$, all tests two-sided). Anecdotal evidence from prior tests suggests that the tuned-search controller performs poorly in real conditions, in part because its model does not adequately deal with partial observability.

We also tested the reinforcement-learning controller’s sensitivity to initial conditions. For 12 initial conditions, we ran 125 trials from

perturbations of a start position, and measured the average distance between balloons (see Methods). The reinforcement-learning controller is much less sensitive to perturbations (56 km average pairwise distance) than StationSeeker (117 km). Supplementary Videos 1–2 illustrate this point in easier and more challenging station-keeping conditions.

Deployment

We used the reinforcement-learning controller to station Loon balloons near an equatorial location over the Pacific Ocean where StationSeeker already performs superbly (see Methods for details on how controllers are integrated into the broader fleet management system). We cumulated a total of 2,884 flight hours from 17 December 2019 to 25 January 2020 (Extended Data Figs. 1–3). Flight data was divided into 851 three-hour periods, each of which can be viewed as a roughly independent sample. This process was repeated with data from 7,475 flight hours concurrently acquired with StationSeeker at the same location (48 segments, 2,185 periods). Real conditions differ from simulation due to modelling discrepancies in flight dynamics and power consumption, making this experiment a major challenge for the reinforcement-learning controller. Additionally, the controllers received observations from multiple other balloons operating in the vicinity of the station. As a result, many of the states encountered during deployment had never been experienced in training (Supplementary Video 3 visualizes a balloon station-keeping over the course of seven days).

Overall, our controller spends more time in range of the station (TWR50 79% versus 72%; $U = 850,410.5$, $P < 10^{-4}$), and uses less power for altitude control (average 29 W versus 33 W, $U = 1,048,814$, $P < 10^{-4}$). The reported TWR50 values are higher than the average for the simulation

benchmark, which is curated to contain an artificially high proportion of difficult conditions. We find that our controller spends far fewer 3-h periods outside of the 50-km range, resulting in a lower 25th-percentile TWR50 (72% versus 39%; Fig. 4a). On the basis of our simulation results, we expect a larger performance gap at other locations.

Within range, StationSeeker attempts to simultaneously slow down and navigate towards the station, a behaviour that often takes it through the centre of the range. By contrast, the reinforcement-learning controller uses different strategies depending on wind conditions. For example, it takes advantage of lateral winds to station-keep without closing in, benefiting from a larger area and thus remaining passive more often. This results in more time spent within the 25–50 km outer ring (Fig. 4b), where it uses significantly less energy (27 W versus 43 W, $U = 287,887$, $n_1 = 812$, $n_2 = 565$, $P < 10^{-4}$; Fig. 4d), Power consumption is also reduced in the inner ring (0–25 km; 15 W versus 18 W, $U = 54,176$, $n_1 = 778$, $n_2 = 116$, $P = 0.0005$).

Outside the range (50–100 km), the reinforcement-learning controller actively works to return within the target region, resulting in shorter excursion lengths (average 3.3 h versus 5.0 h, $U = 21,738$, $n_1 = 295$, $n_2 = 123$, $P = 0.001$; Fig. 4c). This results in increased power consumption beyond 50 km (55 W versus 39 W, $U = 14,060.5$, $n_1 = 367$, $n_2 = 95$, $P = 0.0036$), evidencing how the reinforcement-learning controller utilizes energy savings made while within range. Finally, we find evidence that the controller uses altitude to convert solar energy in excess of battery capacity into potential energy (Fig. 4e). This occurs because ascending does not require power; the stored potential energy is then released early in the night, when power availability is most uncertain.

Discussion

We used reinforcement learning to create a flight controller that can efficiently station a stratospheric balloon for weeks at a time. We dealt with the unavailability of a tractable physical simulator or affordable data-collection mechanism by augmenting an existing meteorological dataset with procedural noise to create simulations sufficient for our purposes. We address the problem of partial observability^{9,38} by incorporating uncertainty variables as additional inputs to the neural network, and used a relative encoding of the wind column to reduce the gap between simulation and reality^{6,7}.

Although short-term, low-level control of the balloon can be achieved using standard linearization techniques², reinforcement learning is particularly well equipped to deal with the discrete factors that must be considered when optimizing complex, long-term objectives such as station-keeping. The algorithmic framework we used is by now reasonably standard, and many state features were inherited from StationSeeker. Reinforcement learning also compares favourably to runtime planning: offloading the optimization process to a training phase enables a longer decision-making horizon and reduces the computational cost of decisions during deployment. Although we reported on a single, controlled experiment, our method has now replaced StationSeeker across the Loon fleet.

In designing intelligent agents that carry out tasks autonomously in the real world, we encounter the issue of domain boundaries—for instance, what a plate-grasping robot²⁹ does once its dishwasher is empty. Currently, in between bouts of cognition, scripted behaviours and engineers take over from the agent to reset the system to its initial configuration, but this reliance on external agency limits the agent's autonomy. By contrast, station-keeping offers an example of a fundamentally continual and dynamic activity, one in which ongoing intelligent behaviour is a consequence of interacting with a chaotic outside world. By reacting to its environment instead of imposing a model upon it, the reinforcement-learning controller gains a flexibility that enables it to continue to perform well over time. In our pursuit of autonomous intelligence, we may do well to pay attention to emergent properties of these and other agent–environment interactions^{49,50}.

Online content

Any methods, additional references, Nature Research reporting summaries, source data, extended data, supplementary information, acknowledgements, peer review information; details of author contributions and competing interests; and statements of data and code availability are available at <https://doi.org/10.1038/s41586-020-2939-8>.

- Lally, V. E. *Superpressure Balloons for Horizontal Soundings of the Atmosphere* Technical report (National Center for Atmospheric Research, 1967).
- Anderson, B. & Moore, B. J. *Optimal Control: Linear Quadratic Methods* (Prentice-Hall, 1989).
- Camacho, E. F. & Bordons, C. *Model Predictive Control* (Springer, 2007).
- Bellman, R. E. *Dynamic Programming* (Princeton Univ. Press, 1957).
- Sutton, R. S. & Barto, A. G. *Reinforcement Learning: An Introduction* 2nd edn (MIT Press, 2018).
- Jakobi, N., Husbands, P. & Harvey, I. Noise and the reality gap: the use of simulation in evolutionary robotics. In *Proc. European Conf. Artificial Life* (eds Moran, F. et al.) 704–720 (Springer, 1995).
- Tobin, J. et al. Domain randomization and generative models for robotic grasping. In *Proc. Intl Conf. Intelligent Robots and Systems* 3482–3489 (IEEE, 2018).
- Levine, S., Kumar, A., Tucker, G. & Fu, J. Offline reinforcement learning: tutorial, review, and perspectives on open problems. Preprint at <https://arxiv.org/abs/2005.01643> (2020).
- Kaelbling, L. P., Littman, M. L. & Cassandra, A. R. Planning and acting in partially observable stochastic domains. *Artif. Intell.* **101**, 99–134 (1998).
- Tesauro, G. Temporal difference learning and TD-Gammon. *Commun. ACM* **38**, 58–68 (1995).
- Mnih, V. et al. Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).
- Silver, D. et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).
- Lauer, C. J., Montgomery, C. A. & Dietterich, T. G. Managing fragmented fire-threatened landscapes with spatial externalities. *For. Sci.* **66**, 443–456 (2020).
- Simão, H. P. et al. An approximate dynamic programming algorithm for large-scale fleet management: a case application. *Transport. Sci.* **43**, 178–197 (2009).
- Mannion, P., Duggan, J. & Howley, E. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In *Autonomic Road Transport Support Systems* (eds McCluskey, T. L. et al.) 47–66 (Springer, 2016).
- Mirhoseini, A. et al. Chip placement with deep reinforcement learning. Preprint at <https://arxiv.org/abs/2004.10746> (2020).
- Nevmyvaka, Y., Feng, Y. & Kearns, M. Reinforcement learning for optimized trade execution. In *Proc. Intl Conf. Machine Learning* (eds Cohen, W. W. & Moore, A.) 673–680 (ACM, 2006).
- Pineau, J., Bellemare, M. G., Rush, A. J., Ghizaru, A. & Murphy, S. A. Constructing evidence-based treatment strategies using methods from computer science. *Drug Alcohol Depend.* **88**, S52–S60 (2007).
- Anderson, R. N., Boulanger, A., Powell, W. B. & Scott, W. Adaptive stochastic control for the smart grid. *Proc. IEEE* **99**, 1098–1115 (2011).
- Glavic, M., Fonteneau, R. & Ernst, D. Reinforcement learning for electric power system decision and control: past considerations and perspectives. *IFAC PapersOnLine* **50**, 6918–6927 (2017).
- Theocharous, G., Thomas, P. S. & Ghavazamdeh, M. Personalized ad recommendation systems for life-time value optimization with guarantees. In *Proc. Intl Joint Conf. Artificial Intelligence* (eds Yang, Q. & Wooldridge, M.) 1806–1812 (AAAI Press, IJCAI, 2015).
- le, E. et al. SlateQ: a tractable decomposition for reinforcement learning with recommendation sets. In *Proc. Intl Joint Conf. Artificial Intelligence* (ed. Kraus, S.) 2592–2599 (IJCAI, 2019).
- Ross, S., Gordon, G. & Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proc. 14th Intl Conf. Artificial Intelligence and Statistics*, (eds Gordon, G. et al.) 627–635 (PMLR, 2011).
- Tan, J. et al. Sim-to-real: learning agile locomotion for quadruped robots. In *Proc. Robotics: Science and Systems XIV* (eds Kress-Gazir, H. et al.) 10 (2018).
- Ng, A. Y., Kim, H. J., Jordan, M. I. & Sastry, S. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems 16 (NIPS 2003)* (eds Saul, L. K. et al.) 799–806 (2004).
- Abbeel, P., Coates, A., Quigley, M. & Ng, A. Y. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems 19 (NIPS 2006)* (eds Schölkopf, B. et al.) 1–8 (MIT Press, 2007).
- Reddy, G., Wong-Ng, J., Celani, A., Sejnowski, T. J. & Vergassola, M. Glider soaring via reinforcement learning in the field. *Nature* **562**, 236–239 (2018).
- Lange, S., Riedmiller, M. & Voigtländer, A. Autonomous reinforcement learning on raw visual input data in a real world application. In *Proc. Intl Joint Conf. Neural Networks* <https://doi.org/10.1109/IJCNN.2012.6252823> (IEEE, 2012).
- Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J. & Quillen, D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *Int. J. Robot. Res.* **37**, 421–436 (2018).
- Kalashnikov, D. et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Proc. Conf. Robot Learning* Vol. 87 (eds Billard, A. et al.) 651–673 (PMLR, 2018).
- Andrychowicz, O. M. et al. Learning dexterous in-hand manipulation. *Int. J. Robot. Res.* **39**, 3–20 (2020).
- Zhang, C. Madden–Julian Oscillation. *Rev. Geophys.* **43**, RG2003 (2005).
- Domeisen, D. I., Garfinkel, C. I. & Butler, A. H. The teleconnection of El Niño Southern Oscillation to the stratosphere. *Rev. Geophys.* **57**, 5–47 (2018).

34. Baldwin, M. et al. The quasi-biennial oscillation. *Rev. Geophys.* **39**, 179–229 (2001).
35. Friedrich, L. S. et al. A comparison of Loon balloon observations and stratospheric reanalysis products. *Atmos. Chem. Phys.* **17**, 855–866 (2017).
36. Coy, L., Schoeberl, M. R., Pawson, S., Candido, S. & Carver, R. W. Global assimilation of Loon stratospheric balloon observations. *J. Geophys. Res. D* **124**, 3005–3019 (2019).
37. Rasmussen, C. E. & Williams, C. K. I. *Gaussian Processes for Machine Learning* (MIT Press, 2006).
38. Sondik, E. *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, Stanford Univ. (1971).
39. Hersbach, H. et al. The ERA5 global reanalysis. *Q. J. R. Meteorol. Soc.* **146**, 1999–2049 (2020).
40. Perlin, K. An image synthesizer. *Comput. Graph.* **19**, 287–296 (1985).
41. Bellemare, M. G., Naddaf, Y., Veness, J. & Bowling, M. The Arcade Learning Environment: an evaluation platform for general agents. *J. Artif. Intell. Res.* **47**, 253–279 (2013).
42. Kolter, Z. J. & Ng, A. Y. Policy search via the signed derivative. In *Proc. Robotics: Science and Systems V* (eds Trinkle, J. et al.) 27 (MIT Press, 2009).
43. Levine, S. & Koltun, V. Guided policy search. In *Proc. Intl Conf. Machine Learning* Vol. 28-3 (eds Dasgupta, S. & McAllester, D.) 1–9 (ICML, 2013).
44. Lin, L. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Mach. Learn.* **8**, 293–321 (1992).
45. Nair, V. & Hinton, G. E. Rectified linear units improve restricted Boltzmann machines. In *Proc. 27th Intl Conf. Machine Learning* (ed. Fürnkranz, J.) 807–814 (ICML, 2010).
46. Dabney, W., Rowland, M., Bellemare, M. G. & Munos, R. Distributional reinforcement learning with quantile regression. In *Proc. AAAI Conf. Artificial Intelligence* 2892–2901 (AAAI Press, 2018).
47. Mnih, V. et al. Asynchronous methods for deep reinforcement learning. In *Proc. Intl Conf. Machine Learning* Vol. 48 (eds Balcan, M.-F. & Weinberger, K. Q.) 1928–1937 (ICML, 2016).
48. Munos, R. From bandits to Monte-Carlo tree search: the optimistic principle applied to optimization and planning. *Found. Trends Mach. Learn.* **7**, 1–129 (2014).
49. Gibson, J. J. *The Ecological Approach to Visual Perception* (Taylor & Francis, 1979).
50. Brooks, R. Elephants don't play chess. *Robot. Auton. Syst.* **6**, 3–15 (1990).

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© The Author(s), under exclusive licence to Springer Nature Limited 2020

Methods

Platform

The Loon superpressure balloons¹ (<http://www.loon.com>) used in our experiments are around 1,800 m³ and carry a payload of around 100 kg. They fly in the stratosphere above wildlife, weather and most planes, using lighter-than-air lift gas to provide buoyancy. They stabilize at the altitude where the force of gravity on the system is equal to the lift created by the buoyancy of the lift gas. Each balloon has two chambers: one that contains lift gas and another that can be filled with ambient air. Doing so changes the mass of the balloon system and causes it to descend. Pumping air into the chamber requires energy but releasing it does not, creating asymmetry in control dynamics.

A fleet dispatch system assigns each balloon to a particular location, and a navigation controller brings it in the vicinity of that location before switching control to the station-keeping controller. All controllers run within a safety layer to ensure important behavioural invariants are respected. As examples, a controller is not permitted to use power in a way that would leave the balloon unable to maintain safety-critical functions, or permitted to navigate to an altitude that would damage the system. The observation vector contains variables indicating whether the balloon has reached the limits of its altitude range and whether it can use energy. These safety limits are present both in training and on the real system, so the algorithm learns to work within the constraints of the safety layer. At the end of their flight, the balloons are navigated to planned locations and landed in coordination with air traffic control.

The controllers navigate by providing a periodically updated setpoint to the altitude-control system. The actions used by the reinforcement-learning controller (ascend, descend and stay) correspond to setpoints at the lowest-permitted, highest-permitted and current altitudes.

Station-keeping as a control problem

Station-keeping is initiated up to 300 km from the station, and continues until the balloon is reassigned. The objective of station-keeping is to maximize the proportion of time spent within range from the moment that the process is initiated. We further require that a controller's average power usage be no greater than StationSeeker's.

The nature of station-keeping creates challenges for standard model-predictive control tools³. There is a nontrivial and nonlinear relationship between decisions and the signal we wish to control—that is, the distance to the station. At night, for example, the controller must weigh making short-term gains against long-term losses due to low battery charge. Because wind uncertainty cannot be accurately described in closed form, there is also no simple way of modelling the consequences of exploration. More generally, the mapping from a state to an optimal decision varies nonlinearly in terms of the input variables. Search-based model-predictive solutions such as the OPD controller are a more natural fit, but are computationally impractical.

Conditions for station-keeping

We say that a location has diverse winds if we can identify altitudes at which the winds blow in opposing directions. We use wind diversity to characterize the difficulty of station-keeping at a particular location, and to filter out impossible training scenarios. In aggregate, we estimate that diverse winds are available in the tropics 67% of time. We obtain this estimate using a 1°-spaced lattice over the tropics (approximately 18,000 locations). For each location, wind diversity is estimated by querying the simulator every hour from 2000 to 2019 and over five random seeds. Each query results in a 'diverse-or-not' outcome. The aggregate availability is the proportion of queries that are diverse. Using the same technique, we estimate at 83.3% the average January availability for the equatorial location of our experiment. Note that, as this number measures the occurrence of a specific wind pattern

and ignores partial observability, it is not directly comparable to TWR50.

The diversity of winds in the tropical stratosphere is influenced by several phenomena. The Quasi-Biennial Oscillation is a series of alternating easterly and westerly winds with a period of approximately 28 months, and is driven by vertically propagating gravity waves originating in the troposphere³⁴. The Madden-Julian Oscillation (MJO) is a progression of convective activity from the Indian Ocean into the Pacific Ocean with a periodicity of 30–90 days³². It has been shown that the movement of precipitation centres by the Madden-Julian Oscillation alters the east-west winds in the tropical stratosphere³¹. Finally, the El Niño–Southern Oscillation influences the tropical circulation in both the troposphere and stratosphere through several mechanisms³³.

StationSeeker

StationSeeker assigns a score s_l to pressure levels, l , ranging from 5 kPa to 14 kPa in 50 Pa increments ($l=1, \dots, 181$). The highest-scoring altitude is designated as a setpoint. The score function is parametrized to balance four considerations within the station-keeping objective: reaching the station, remaining close to the station, exploring the wind field and limiting power consumption.

A wind score g_l is computed for each pressure level l on the basis of the wind magnitude μ_l and bearing θ_l :

$$g_l = (1 - \alpha_\Delta) e^{-\omega_\Delta \theta_l} + \alpha_\Delta e^{-k_1 \mu_l}.$$

The term $\alpha_\Delta \in [0, 1]$ is the relative weight associated with the wind magnitude and bearing, and the term ω_Δ defines the magnitude of the penalty associated with winds whose direction makes a large angle with the direction of the station. Both of these decrease with the distance to the station, Δ , and k_1 is a constant. Close to the station, the magnitude term dominates and the controller seeks slow winds.

To incentivize exploration and minimize power consumption, s_l interpolates between g_l and a default score g_{unknown} and penalizes large altitude changes:

$$s_l = (1 - u_l) g_l + u_l g_{\text{unknown}} + k_2 e^{-k_3 |l - l_{\text{current}}|}.$$

The interpolation factor $u_l \in [0, 1]$ is the uncertainty at pressure level l , and k_2 and k_3 are constants. Interpolating using uncertainty leads the controller to navigate to unexplored altitudes when no high-scoring winds have been observed.

All parameters effect a trade-off between power and performance; they were previously tuned through multiple interactions with the simulator and manual analysis of real flight data. We validated this tuning by evaluating different settings of ω_Δ , k_1 , k_2 and g_{unknown} on our 6,000-flight benchmark. We sampled 50 settings uniformly at random from reasonable ranges, and additionally varied one parameter at a time (Extended Data Fig. 4). At equal or lower power consumption, the best parametrization of the evaluation function achieves barely higher TWR50 than StationSeeker (41.6% versus 40.5%), within the range where the two can be considered equivalent.

Tree-search controller (OPD)

In theory, a search-based approach can achieve near-optimal performance by enumerating all possible futures. However, using it in a production-grade flight controller is computationally impractical as it requires growing the search tree at every decision point. This is problematic in our setting, where partial observability creates sizeable randomness.

Instead, the tree-search controller models the flight dynamics deterministically, using first-order physics to determine the balloon's position following a decision. It models night-time power consumption as a constant baseline rate plus an additional descent cost. Although practical, one downside of this simplified model is that

Article

it cannot account for measurements obtained during the planning process.

We use OPD⁴⁸ to plan over this model. OPD is designed for deterministic discounted decision processes. It improves over breadth-first search by focusing on the most promising sequences first, enabling longer planning horizons. This is particularly effective in our setting, where many decisions move the balloon away from the station. Once N nodes have been expanded, OPD selects the action offering the greatest lower bound on the return.

Gaussian process queries dominate the computational cost of a node expansion (75–95% of total cost). To make the process tractable we use a caching mechanism that aggregates neighbouring winds and impose a planning limit of 500 km and 33 h. By parallelizing the process over 8 threads, our algorithm can expand 200,000 nodes in roughly 800 ms. Although additional engineering could scale this process further, our results suggest that this would produce, at best, marginal gains. The search process is parametrized by the duration T of a single decision and a stride parameter S . The available setpoints are those reachable within that time interval, spaced S pascals apart. If $T = 15$ min and $S = 100$ Pa these are $0, \pm 100$ and ± 200 Pa, relative to the current altitude. Planning at the same resolution as the reinforcement-learning controller ($T = 3$ and $S = 50$) remains computationally infeasible; as a nontrivial point of comparison, even using $T = 15$ and $S = 100$ achieves a low TWR50 of 34.4%.

The search controllers reported in Fig. 3b were obtained by varying $T \in [15, 60]$, $S \in [50, 200]$, $\gamma \in \{0.993, 0.999\}$, and a potential penalty for power usage ($f_\omega \in \{0.9, 1.0\}$; see below). The best controller uses $T = 60$, $S = 100$, $\gamma = 0.993$ and $f_\omega = 1.0$.

The tuned-search controller replaces the tree by a lattice of 400-km² cells. All locations within a cell are approximated with its centre. Additionally, it uses the negative of the distance to the station as reward, ignoring power or the 50-km boundary.

Reward function

We consider a distance-based reward

$$r_{\text{dist}}(\Delta) = \begin{cases} 1.0 & \text{if } \Delta < \rho \\ c_{\text{cliff}} 2^{-(\Delta-\rho)/\tau} & \text{otherwise} \end{cases},$$

where Δ is the distance to the station. Smaller values of c_{cliff} penalize exiting the range so that the objective is preserved; with $c_{\text{cliff}} = 1$ and a low decay rate τ , the balloon may prefer to remain just outside the range, owing to discounting and approximation errors. The best performance is provided by $\tau = 100$ km and $c_{\text{cliff}} = 0.4$ (Fig. 2a).

We modulate this reward by a multiplicative penalty for energy that is drawn from the battery or could be used to charge it. We found this performs better than an additive penalty, which dominates the action-value estimates far from the station. We use a normalized measure of power consumption $\omega \in [0, 1]$ during the 3-min time step. The power-penalized reward is $r(s, a) = r(\Delta, \omega) = f_\omega r_{\text{dist}}(\Delta)$, with

$$f_\omega = \begin{cases} 0.95 - 0.3\omega & \text{if } \omega > 0 \\ 1.0 & \text{otherwise} \end{cases}.$$

State encoding

The state vector consists of 1,083 wind variables (361 triples) and 16 ambient variables (Extended Data Table 1). 181 triples correspond to pressure levels from 5 kPa to 14 kPa in 50-Pa increments, and the central triple always corresponds to the balloon's current altitude. The remaining triples are invalid for any given state; some altitudes are also inaccessible owing to changes in temperature, infrared radiation and levels of lift gas. We encode these using the limit triple (1, 1, 0), semantically equivalent to being maximally confident that the wind is blowing infinitely fast away from the station.

Deep reinforcement learning

An action value⁵² describes the expected return that follows from taking action a in state s :

$$R_{s,a} = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right].$$

We estimate action values using the QR-DQN algorithm⁴⁶, an open-source implementation of which is available online⁵³. QR-DQN uses collections of N real-valued locations $\theta_{s,a}^j$:

$$R_{s,a} \approx \frac{1}{N} \sum_{j=1}^N \theta_{s,a}^j$$

Each location approximates a quantile of the probability distribution of the random return. This approach stabilizes learning in neural networks and improves performance compared to direct mean estimation⁵⁴; Fig. 2b shows a small (+1.2%) TWR50 improvement. We also use these locations to estimate the maximum performance in our simulation benchmark (see below).

We model QR-DQN's locations using a seven-layer, 600-unit neural network. The network takes in a state vector as input and outputs 51 quantiles per action. Fewer (21) or more (101) quantiles did not affect performance. We maintain a separate target network⁴¹ and use n -step backups⁵. The controller then acts by selecting the action with the highest estimate. The network weights are optimized using Adam⁵⁵.

Extended Data Table 2 summarizes the parameters of the learning algorithm (hyperparameters). These were tuned using Vizier⁵⁶ and grid search. The learning process is most sensitive to Adam parameters, and these formed the bulk of early tuning. The process otherwise performs well for a range of parameters and many of these (for example, mini-batch size, update horizon and number of quantiles) are imported from open-source implementations. The deployed controller was trained for 24 days (wall-clock time) and 1.1 billion gradient updates using a V100 graphics processing unit (GPU), and was selected from among ten similarly trained parametric variations. The controllers with performances reported in Fig. 1 were trained in the same fashion, but for 300 million gradient updates.

Training-data generation

A trial is defined by the balloon's initial longitude, latitude, altitude, date, time, station location and a random seed for wind noise. For training, trial start dates and locations are sampled uniformly at random from 2005–2010 (inclusive) and from the tropics (latitudes -25° to 25°); configurations lacking wind diversity are rejected. The balloon's initial altitude is sampled uniformly from the accessible range, and its initial distance from the station is drawn from a scaled beta distribution with support 0–200 km; note that poor navigation will cause the balloon to experience larger distances. This procedure provides a diversity of learning scenarios, as weather patterns recur seasonally but differ across the planet.

Data are recorded as a series of state–action–reward transitions. Once a trial completes, its transitions are appended to a replay buffer chosen uniformly at random. The final transition of a trial lacks continuation and is ignored.

To efficiently explore the consequences of different decisions, we use an exploratory policy. This policy first samples a setpoint uniformly at random from the accessible range. This setpoint is then periodically updated with zero-mean Gaussian noise. This causes the balloon to perform a random walk between altitudes, rather than taking individual actions, providing momentum to exploration and improving its efficiency.

A majority (80% probability) of trials are selected for exploration. These trials interleave the reinforcement-learning controller with the exploratory policy in intervals of four and two hours, respectively, for a total of eight cycles per trial. Interleaving allows the controller to efficiently test macroscopic deviations from its usual behaviour, as following the exploratory policy for more than a few hours would produce mostly uninteresting data.

Simulation benchmark

Our simulation benchmark is divided equally across three locations in Peru and two in Kenya, regions in which Loon operates. This assesses our controller's ability to respond to different weather statistics; empirically, station-keeping is easier over Kenya than over Peru. One hundred dates selected from 2000 to 2015 correspond to each of the five locations, omitting training dates (2005–2010), each with 12 random seeds. This yields 6,000 flight configurations. The per-location dates are sampled in equal proportions from five difficulty categories derived from StationSeeker's performance. Each category corresponds to a 20% TWR50 interval. For example, there are 20 dates for which StationSeeker achieves 0–19% TWR50. The balloon's initial position is sampled as in training.

We simulated 12 more flights sharing a common Peru station, beginning on the first of each month of 2002, which we refer to as 'the 2002 simulations'. This procedure yielded 11,520 states. Of these, the first state of each trajectory is discarded to correct for the simulator's burn-in period.

Estimation of maximum benchmark performance

By providing the OPD controller with perfect wind information, we reduce station-keeping to a path-planning problem with energy constraints. However, partial observability carries a performance cost. This is reflected in the spread of the neural network's return distribution predictions (Extended Data Fig. 5). We therefore estimate the lower limit on maximum performance as the perfect-information upper limit, minus the cost of partial observability.

We assume the following: 1) the action-value distribution predictions are reasonably accurate; 2) the environment dynamics are deterministic given the random seed, such that any predicted quantile is obtained by some sequence of decisions; and 3) the predicted outcomes correspond to a sum of contiguous binary rewards. This last assumption enables us to interpret action-value differences as differences in the time spent successfully station-keeping. We write

$$R_{s,a} = \gamma^{T_{s,a}},$$

where $T_{s,a}$ is the number of time steps before the balloon arrives within range of the station, after which point it remains within range until the simulation ends. An alternative model, $R_{s,a} = (1 - \gamma^{T_{s,a}+1})/(1 - \gamma)$, leads to similar estimates.

Under these assumptions, the only source of randomness is partial observability. The largest predicted quantile, $R_{s,a}^*$, therefore estimates the optimal return in the absence of partial observability. We associate this larger quantile with an earlier arrival time, $T_{s,a}^*$. For a given state s , we compute the time steps saved owing to full observability:

$$T_{s,a}^* - T_{s,a} = \frac{\log(\max_a R_{s,a}^*) - \log(\max_a R_{s,a})}{-\log \gamma}.$$

Using the 2002 simulations, we estimate $T_{s,a}^* - T_{s,a} = 57$ steps, or 2.85 h. We convert this difference into TWR50 by dividing it by 24 h (480 time steps). 24 h corresponds to the expected length of time remaining in a simulated trial, when sampling a time step at random, as we do here. Anecdotally, it also roughly corresponds to the longest time period in which a balloon may station-keep without interruption, owing to changing winds and low battery charge at night. This calculation results in a

TWR50 difference of 11.9%, which, once subtracted from the perfect-information limit, yields 56.8% TWR50 as a plausible maximum on performance on the simulation benchmark.

Sensitivity to initial conditions

For each of the 2002 simulations, we consider 125 perturbations of the balloon's initial position, arranged in a box ($0, \pm 0.01, \pm 0.02$ degrees latitude and longitude, $0, \pm 100, \pm 200$ Pa). We record their average distance to the station and the average distance between pairs of balloons (ignoring altitude), which we call the pairwise distance (Extended Data Fig. 6). The numbers reported in the main text correspond to pairwise distances averaged across months; the medians across months are 55 km (reinforcement-learning controller) and 66 km (StationSeeker).

Response to bearing and magnitude as a function of distance

We use a sensitivity analysis to study the reinforcement-learning controller's decision-making process, focusing on how it trades off magnitude and bearing as a function of distance. We use finite differences to estimate the partial derivative of the controller's action-value estimates with respect to input i —that is, its 'response' to i . For a given state s and action a we query the neural network for its action-value estimate $R_{s,a}$. We then add $\delta = 0.01$ to the component of s corresponding to input i to produce a new state $y(i)$, and query the neural network for the new estimate $R_{y(i),a}$. The network's response e_i is

$$e_i = \frac{R_{y(i),a} - R_{s,a}}{\delta}.$$

We repeat the process for each state s of the 2002 simulations, resulting in an average response \bar{e}_i .

To obtain a response that depends on the distance Δ , we vary the distance-to-station input from 0 to 200 km, and fix it across states before querying the network. This yields a function $\hat{e}_i(\Delta)$, which we rescale according to

$$\hat{e}_i(\Delta) = \frac{\bar{e}_i(\Delta)}{\max_{\Delta'} |\bar{e}_i(\Delta')|}.$$

We compare $\hat{e}_i(\Delta)$ to the derivative of StationSeeker's score function with respect to i , rescaled as above so that the two controllers' responses can be compared. Negative responses to magnitude and bearing respectively indicate an aversion to fast winds and to winds bearing away from the station.

Both controllers emphasize magnitude and bearing respectively near and far from the station (Extended Data Fig. 7). However, the reinforcement-learning controller exhibits a sharp change in sensitivity at 50 km. Beyond this boundary, it prefers fast winds (positive response), supporting our empirical observation that the controller works to return within range.

Preprocessing of real flight data

We divide the flight data from our Pacific Ocean experiment into segments, each corresponding to a specific balloon station-keeping during a contiguous time period. Segments shorter than 6 h are omitted. For our reinforcement-learning controller, this yields 25 segments (mean segment length: 115 h, maximum length: 377 h). For StationSeeker, this yields 48 segments. Extended Data Figs. 1–3 depict all segments lasting at least 12 h. Most segments begin with the balloon approaching from a distance; in our analysis we discard this initial approach so that all flight segments start 50 km from the station.

Data availability

The data analysed in this paper are available from the corresponding authors on reasonable request.

Code availability

The code used to train the flight controllers is proprietary. The code used to analyse the generated data is available from the corresponding authors on reasonable request.

51. Alexander, M., Grimsdell, A., Stephan, C. & Hoffmann, L. MJO-related intraseasonal variation in the stratosphere: gravity waves and zonal winds. *J. Geophys. Res. D Atmospheres* **123**, 775–788 (2018).
52. Watkins, C. J. C. H. *Learning from Delayed Rewards*. PhD thesis, Cambridge Univ. (1989).
53. Castro, P. S., Moitra, S., Gelada, C., Kumar, S. & Bellemare, M. G. Dopamine: a research framework for deep reinforcement learning. Preprint at <https://arxiv.org/abs/1812.06110> (2018).
54. Bellemare, M. G., Dabney, W. & Munos, R. A distributional perspective on reinforcement learning. In *Proc. Intl Conf. Machine Learning* Vol. 70 (eds Precup, D. & Teh, Y. W.) 449–458 (PMLR, 2017).
55. Kingma, D. & Ba, J. Adam: A method for stochastic optimization. In *Proc. Intl Conf. Learning Representations* (eds Benigo, Y. & LeCun, Y.) (2015).
56. Golovin, D. et al. Google Vizier: a service for black-box optimization. In *Proc. ACM SIGKDD Intl Conf. Knowledge Discovery and Data Mining* (eds Matwin, S. et al.) 1487–1496 (ACM, 2017).

Acknowledgements We are grateful to J. Davidson for early ideation and prototyping. We thank V. Vanhoucke, K. Choromanski, V. Sindhwani, C. Boutilier, D. Precup, S. Mourad, S. Levine, K. Murphy, A. Faust, H. Larochelle and J. Platt for discussions; M. Bowling, A. Guez, D. Tarlow, J. Drouin, and M. Brenner for feedback on earlier versions of the manuscript; W. Dabney for feedback and help with design; T. Larivee for help with visuals; N. Mainville for project management support; R. Carver for information on weather phenomena; and the Loon operations team.

Author contributions S.C. conceptualized the problem. J.G., S.C., S.S.P., P.S.C. and S.M. built the technical infrastructure. S.C., M.G.B., J.G., M.C.M. and P.S.C. developed and tested the algorithm. M.C.M., M.G.B., P.S.C., S.C., S.S.P., J.G. and Z.W. performed experimentation and data analysis. M.G.B. and S.C. managed the project. M.G.B., S.C., M.C.M., P.S.C. and S.S.P. wrote the paper. Authors are listed alphabetically by surname.

Competing interests M.G.B., S.C., J.G. and M.C.M. have filed patent applications relating to navigating aerial vehicles using deep reinforcement learning. The remaining authors declare no competing financial interests.

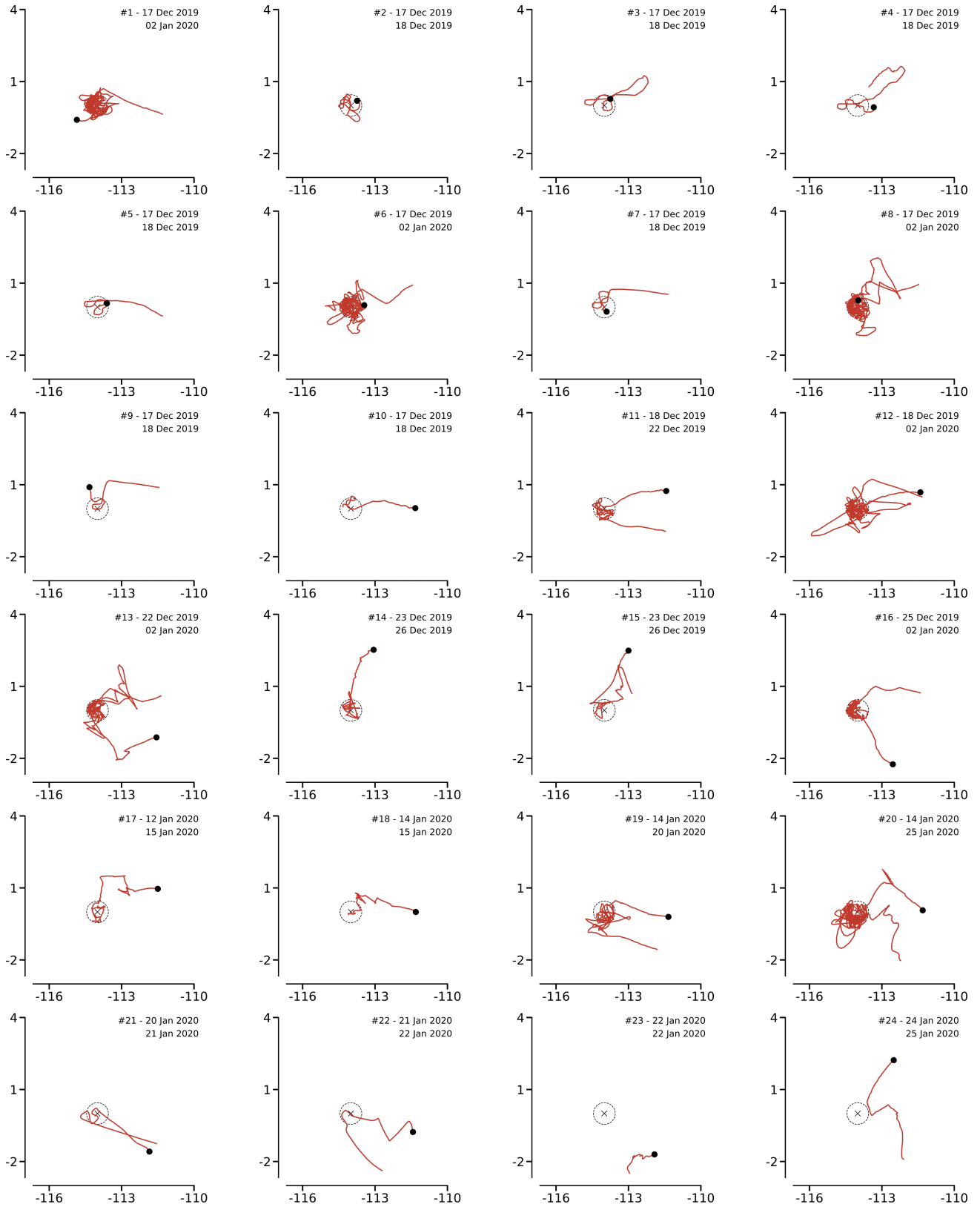
Additional information

Supplementary information is available for this paper at <https://doi.org/10.1038/s41586-020-2939-8>.

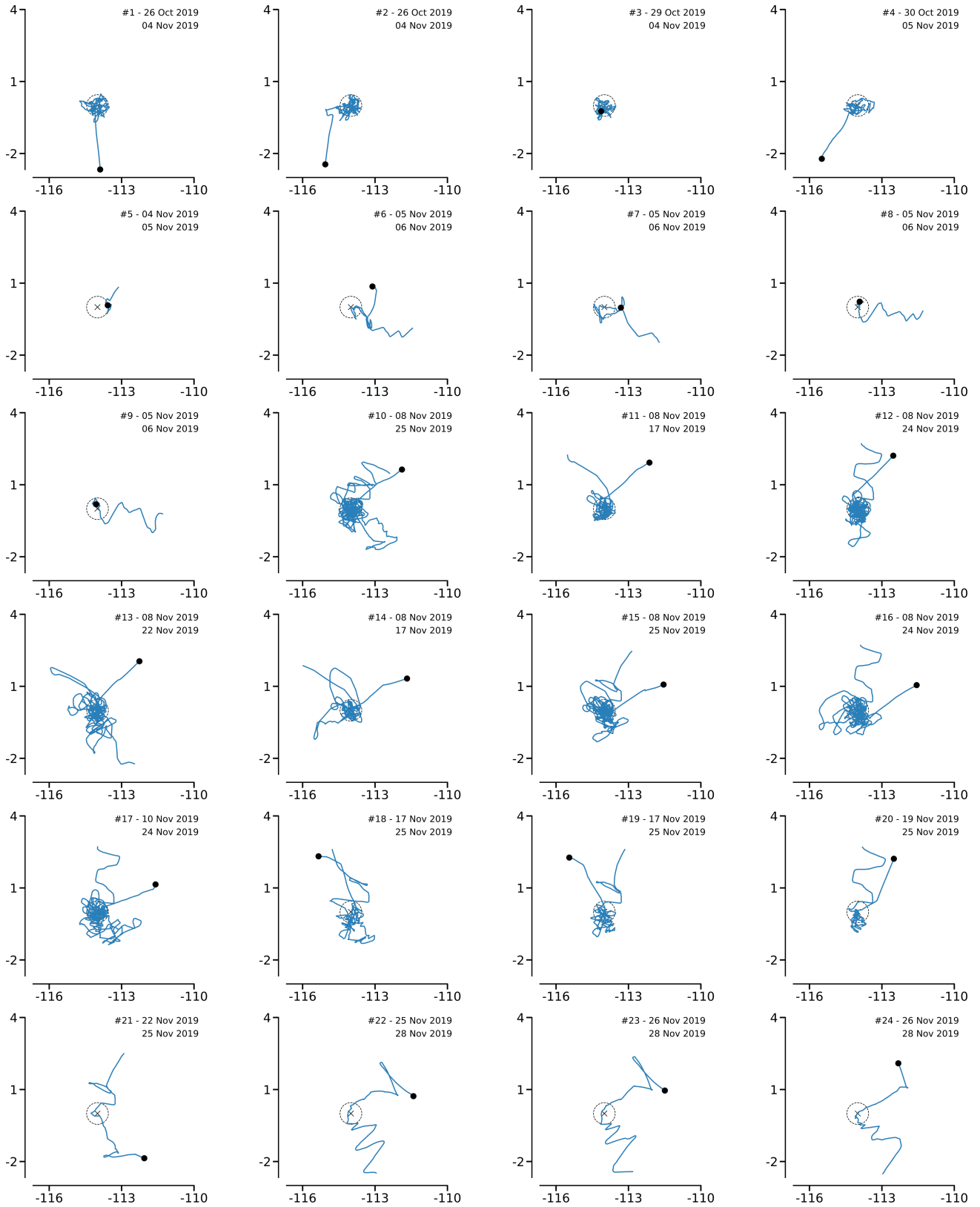
Correspondence and requests for materials should be addressed to M.G.B. or S.C.

Peer review information *Nature* thanks Scott Osprey and the other, anonymous, reviewer(s) for their contribution to the peer review of this work. Peer reviewer reports are available.

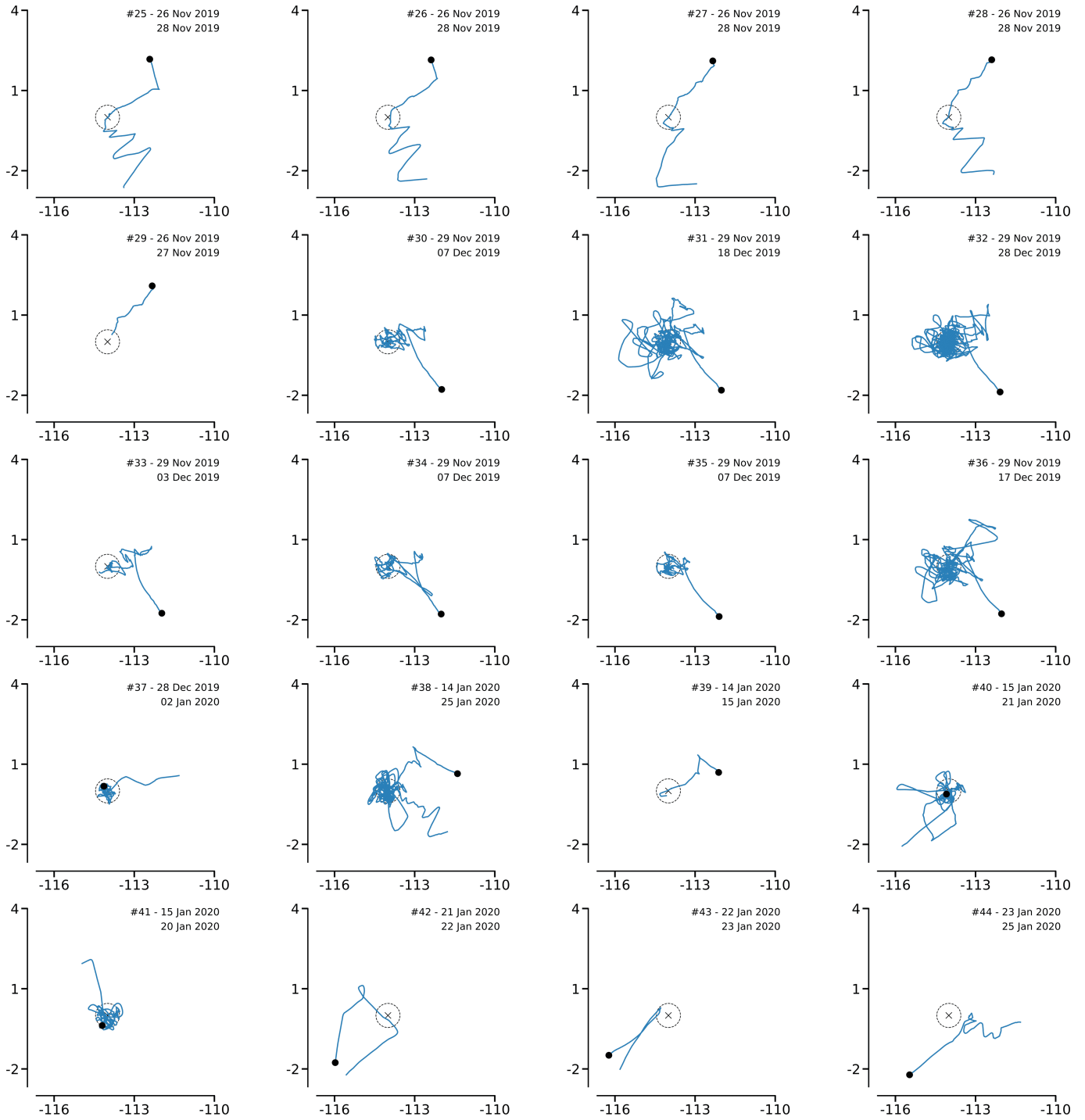
Reprints and permissions information is available at <http://www.nature.com/reprints>.



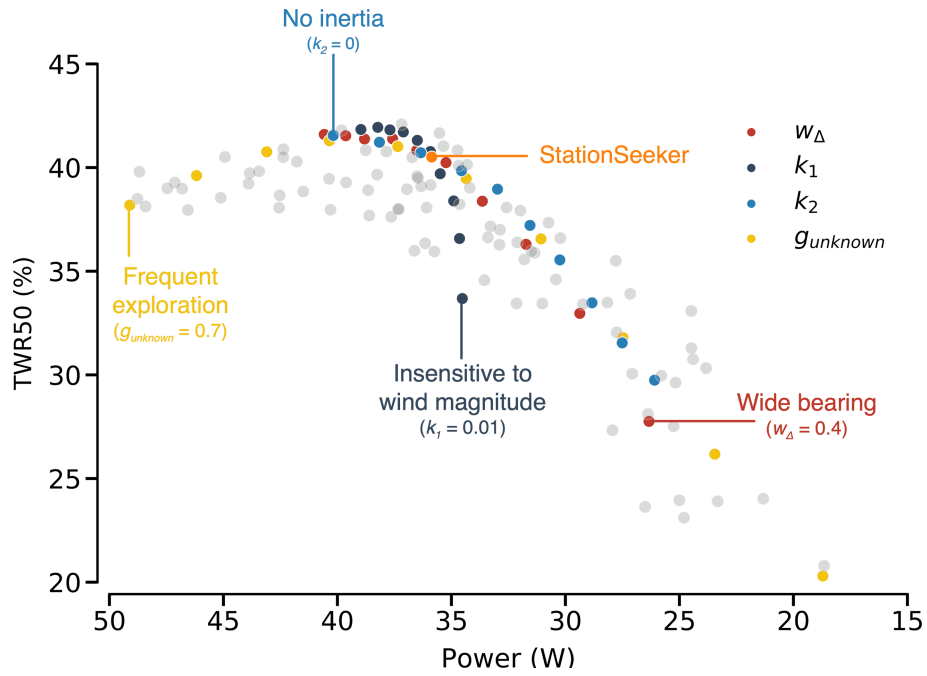
Extended Data Fig. 1 | Flight paths of the reinforcement-learning controller during the Pacific Ocean experiment. The x and y axes represent longitude and latitude.



Extended Data Fig. 2 | Flight paths of StationSeeker during the Pacific Ocean experiment, 1 of 2. The x and y axes represent longitude and latitude, respectively.

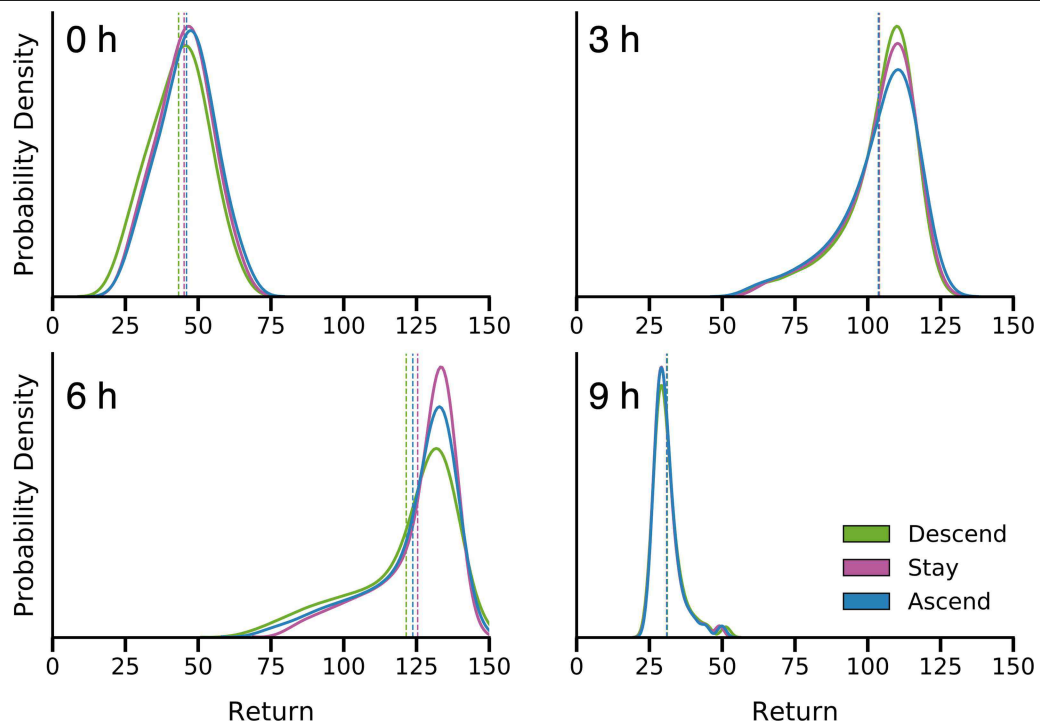


Extended Data Fig. 3 | Flight paths of StationSeeker during the Pacific Ocean experiment, 2 of 2. The x and y axes represent longitude and latitude, respectively.



Extended Data Fig. 4 | TWR50 and power consumption for different parametrizations of StationSeeker's score function. Grey points indicate settings chosen uniformly at random from the following ranges: $w_\Delta \in [0.4, 0.8]$

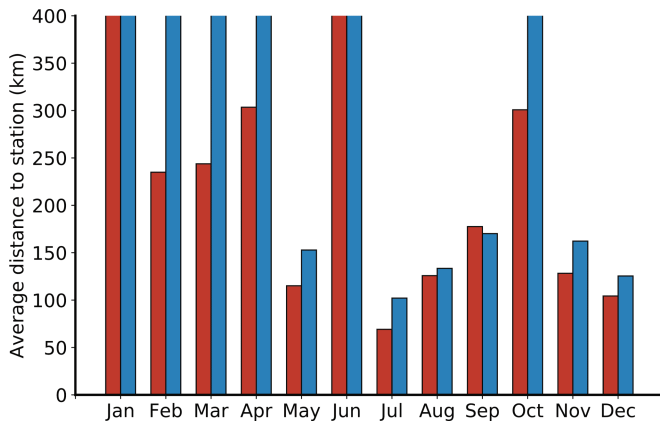
(at close range), $k_1 \in [0.01, 0.15]$, $g_{\text{unknown}} \in [0.4, 0.6]$ and $k_2 \in [0, 0.2]$. Each parameter was also varied in isolation (coloured points). Semantically interesting parameter choices are highlighted.



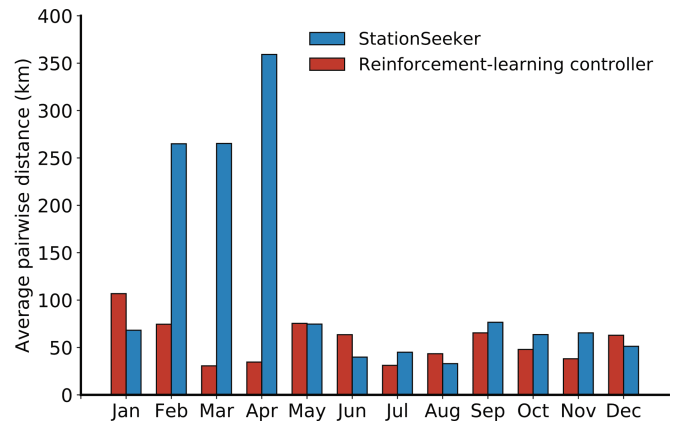
Extended Data Fig. 5 | Distribution of returns predicted by the neural network. Each panel indicates the predicted distributions for a particular state and action. The 51 quantiles output by the network are smoothed using kernel density estimation (σ determined from Scott's rule with interquartile range

scaling). The dashed lines indicate the average of these locations. The states with depicted distributions are from different times (0, 3, 6 and 9 h) into the July 2002 simulation. We use the largest quantile to estimate the return that could be realized in the absence of partial observability.

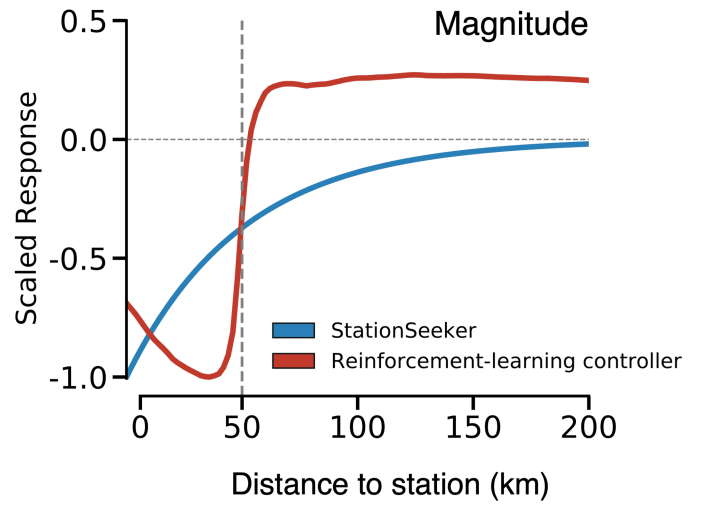
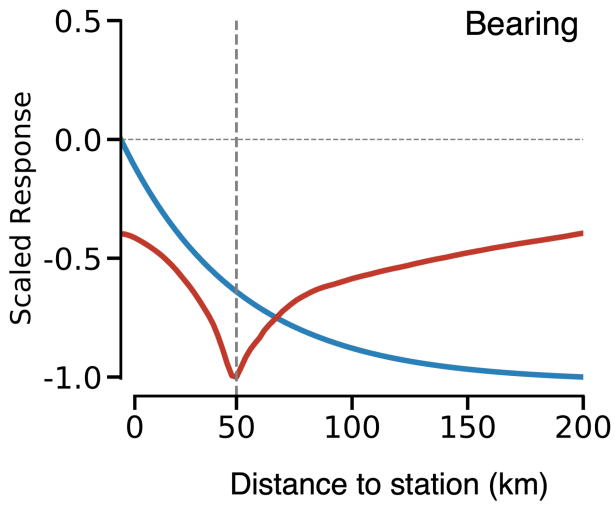
Article



Extended Data Fig. 6 | Average distances and pairwise distances for perturbations of 12 initial conditions. a. Distance to station, averaged over 125 perturbations. These numbers highlight how the 1 January to 1 June 2002 simulations (May excluded) were challenging station-keeping conditions.



The 1 January configuration, in particular, lacked wind diversity. **b.** Average distance between pairs of balloons (7,750 pairs). Our controller exhibits greater robustness to challenging conditions.



Extended Data Fig. 7 | Scaled response of controllers to wind bearing and magnitude as a function of distance. We use the derivative of the network's action-value estimates, or response, as a proxy for the relative weight of an

input. The two inputs tested here are the wind bearing and magnitude at the balloon's altitude; the curves report the derivative for the 'stay' action.

Article

Extended Data Table 1 | Inputs to the flight controller

Feature	Range	Normalized Range	Notes
Altitude of balloon	5 – 14 kPa	[0, 1]	
Battery charge	0 – 100%	[0, 1]	
Solar elevation	-90 – 90°	[-1, 1]	
Distance to station	0 – ∞ km	[0, 1]	Normalized: $f(x) = \frac{x}{x+250}$
Relative bearing	0 – 180°	[-1, 1], [0, 1]	Normalized: $f(\theta) = (\cos \theta, \sin \theta)$
Time from sunrise	0 – 360°	[-1, 1], [-1, 1]	Normalized: $f(\theta) = (\cos \theta, \sin \theta)$
Navigation enabled	Boolean	-	Unary encoding
Has excess energy	Boolean	-	
Descent cost	0 – 300 W	[0, 1]	
Internal pressure ratio	1 – 2	-	
Last command	0, 1, 2	-	Ascend, descend, stay
Wind column (×361)			
Magnitude	0 – ∞ m/s	[0, 1]	Normalized: $f(x) = \frac{x}{x+30}$
Relative bearing	0 – 180°	[0, 1]	
Uncertainty	0 – 100%	[0, 1]	

Unless otherwise stated, inputs are normalized using linear rescaling $f(x) = (x - x_{\min}) / (x_{\max} - x_{\min})$. The last three sets of input values encode the wind column at the balloon's position, centred at the balloon's altitude.

Extended Data Table 2 | Hyperparameters defining the deep reinforcement-learning algorithm

Hyperparameter	Value	Notes
Network layer width	600	
Number of locations	51	Used by QR-DQN
Discount factor	0.993	
Update horizon	5	Used by the n -step method
Minibatch size	32	
Replay buffer size	500,000	
Number of replay buffers	4	
Number of actors	100	
Adam step-size	2e-6	Linearly decayed to 4e-7 in 5 million updates
Adam epsilon	0.00002	
Target network update frequency	100	
Reward function ρ	50	Range (in km) within which $r = 1$
Reward function τ	100	Decay rate (km)
Reward function c_{CLIFF}	0.4	Reward when beyond ρ km