nature machine intelligence

ARTICLES

https://doi.org/10.1038/s42256-020-00279-7

Enhancing optical-flow-based control by learning visual appearance cues for flying robots

G. C. H. E. de Croon[1]✉, C. De Wagter[1] and T. Seidl[2]

Flying insects employ elegant optical-flow-based strategies to solve complex tasks such as landing or obstacle avoidance. Roboticists have mimicked these strategies on flying robots with only limited success, because optical flow (1) cannot disentangle distance from velocity and (2) is less informative in the highly important flight direction. Here, we propose a solution to these fundamental shortcomings by having robots learn to estimate distances to objects by their visual appearance. The learning process obtains supervised targets from a stability-based distance estimation approach. We have successfully implemented the process on a small flying robot. For the task of landing, it results in faster, smooth landings. For the task of obstacle avoidance, it results in higher success rates at higher flight speeds. Our results yield improved robotic visual navigation capabilities and lead to a novel hypothesis on insect intelligence: behaviours that were described as optical-flow-based and hardwired actually benefit from learning processes.

The miniaturization of electronics allows for the creation of tiny flying robots that can navigate in narrow spaces, are cheap enough to be produced in large numbers and are inherently safe for use around humans. To perform economically or societally useful tasks like monitoring plants in a greenhouse or surveying an industrial plant[1], these robots will have to fly completely autonomously. This is a substantial challenge due to the robots' extremely limited payload capabilities. They can only carry small sensors and have highly limited on-board processing capabilities. Flying insects successfully cope with similar restrictions and have thus served as a rich source of inspiration for creating autonomously flying robots[2]. In their turn, the robots can be used as embodied models of flying insects for testing hypotheses that are relevant to biology[3,4].

For achieving autonomous flight of tiny flying robots, most attention has been given to the ingenious ways in which flying insects exploit the visual cue of optical flow[2,5–7]. For the task of landing, for example, honeybees are known to follow a strategy of keeping the optical flow constant[8,9]. Roboticists have tried to reproduce this in flying robots. Specifically, robots equipped with a downward looking camera attempt to keep the ventral flow constant for grazing landings[10] and the divergence constant for straight vertical landings[11–13]. Recently, a few studies have employed an evolutionary robotics approach in the hope of finding neural controllers that can handle this difficult nonlinear control problem[14–16]. For the task of obstacle avoidance, inspiration is drawn from how insects avoid looming visual cues[17–23]. Robots equipped with forward-looking cameras determine when they are about to collide by means of the divergence, which captures the expansion of objects in their view over time[24,25].

Despite the large body of research on optical flow control[26], its success has been limited by two major fundamental shortcomings that arise from the physical nature of optical flow. The first shortcoming is that optical flow only captures the ratio of velocity and distance and does not allow these two quantities to be disentangled. This is referred to as the 'chicken-and-egg' problem by Serres and Ruffier[26], and manifests itself as an ambiguity in the optical flow visual inputs. Figure 1a illustrates that two landing honeybees with different heights z and vertical velocities v_z experience the same visual angular increase to an object underneath over time. The honeybees thus perceive the same optical flow divergence, which is the reason why distance perceived with optical flow is typically termed 'unscaled' or 'relative' with respect to the velocity. This ambiguity is highly problematic for a controller that needs to keep the divergence constant. If the controller does not adapt its optical flow control gain to the height, there is a point during landing where it will start to self-induce oscillations and finally become unstable[27]. Furthermore, suppose that a controller succeeds at keeping the divergence constant. At which moment in time would it then execute its final landing procedure, such as switching off the propellers for a quadrotor or extending the legs for a honeybee? The constant divergence does not provide any useful information on this decision.

The second shortcoming is that optical flow is very small close to the 'focus of expansion' (FoE), leading to difficulties in detecting obstacles in the most important direction, that is, the flight direction[20,22,28]. Figure 1b illustrates a honeybee moving straight forward through its environment. The FoE is the point in its view from which all optical flow originates, and it lies straight ahead of the honeybee on the dashed line. Obstacles further from the FoE (for example, the green obstacle) lead to a bigger angular change over time than obstacles closer to the FoE (for example, the red obstacle). The angular change, and hence optical flow, of obstacles close to the FoE is typically even smaller than the error of the optical flow determination. The consequence is that detecting obstacles is most difficult in the most relevant direction, that is, in the flight direction. This problem is aggravated by the fact that estimation of the FoE image location itself is also difficult, leading to additional noise[28]. The 'FoE problem' is the main reason that most of the obstacle avoidance studies in the robotics literature actually have the robot perform corridor following (for an overview see ref. [26]), eschewing the issue of frontal obstacles. An elegant way to circumvent the FoE problem is to make use of active vision techniques. For example, in ref. [29], a drone moves diagonally up and down, so that the drone can more easily extract unscaled, inverse depth from the optical flow, allowing it to pass through various types of gap. In ref. [30], a

[1]Micro Air Vehicle laboratory, Control and Operations, Faculty of Aerospace Engineering, TU Delft, the Netherlands. [2]Westphalian Institute for Biomimetics, Westphalian University of Applied Sciences, Bocholt, Germany. ✉e-mail: G.C.H.E.deCroon@tudelft.nl

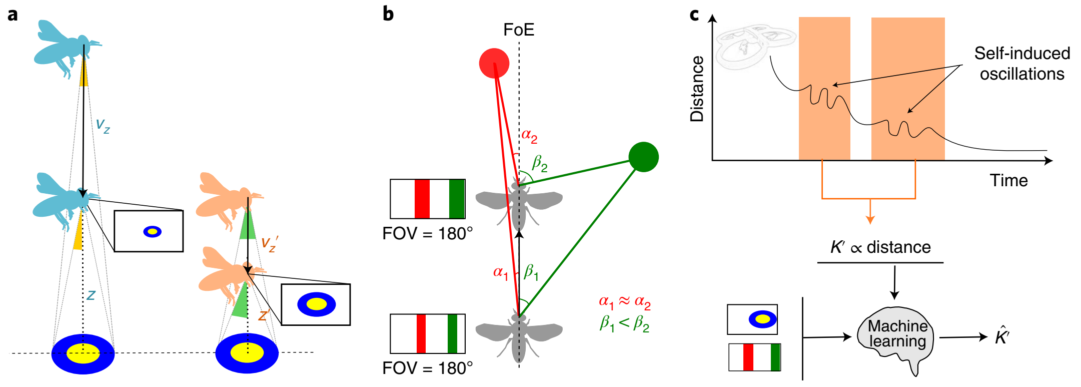


Fig. 1 | Fundamental optical flow problems and the proposed solution. a, Optical flow cannot disentangle distance and velocity. The blue and orange honeybees perceive exactly the same angular increase of the blue-yellow flower (a doubling, as illustrated by the coloured triangles—they cover the full angle in the top position and half the angle in the bottom position). This implies an identical divergence for the two honeybees, although they have a different height and velocity. The two insets show the downward views of the blue and orange honeybees. Although the divergence is identical, the flower is much larger in the orange bee's field of view (FOV). **b**, Top view of a honeybee travelling forward. The FoE is located in the direction of flight. Close to the FoE, the optical flow is so small that it is at a sub-error level, making obstacles in the flight direction difficult to detect. The insets show the forward field of view of the grey honeybee at both time instances. Although the angular rate of the red obstacle is small, its subtended visual angle is as large as that of the green obstacle. **c**, Overview of the proposed learning process that allows a robot to exploit the visual appearance of objects in its environment for improved optical flow control. During self-induced oscillations (orange areas), the distance to the observed object is linearly related to the optical flow control gain K'. The oscillations can be detected by the robot to perceive distance[27]. We propose to use the gain K' in a self-supervised learning process in which the robot learns to associate the visual appearance of images (illustrated by the inset) to the gain K'. After learning, oscillations are no longer necessary, allowing the robot to perceive distances immediately.

simulated bee 'zig-zags' forward, so that the FoE is changing from left to right. In addition, it integrates inverse depth estimates over time, allowing the estimation of unscaled depth also in the average forward flight direction. With these techniques, depth is still unscaled and additional motions always stay necessary.

Self-supervised learning of distance estimation

To overcome both fundamental optical flow problems, we propose to have robots learn the visual appearance of their environment. Figure 1 illustrates how the visual appearance of objects in the environment contains the solution to both problems. In Fig. 1a, the two honeybees have the same optical flow but see a different visual size of the flower beneath. In Fig. 1b, the red obstacle is hard to detect by means of optical flow but easy to detect by means of its visual appearance and size. Although it may be evident that visual appearance can form a valuable complement to optical flow, the main question to solve is 'how can a robot or insect associate the visual appearance and size of an object with distance when optical flow by itself does not contain this information?' The answer lies in a preliminary study of one of the authors[27], in which an active strategy is proposed that allows robots to disentangle distance from velocity by means of adapting control gains during flight. The theoretical analysis in that article showed that, when the optical flow control gain, referred to as K, is adapted to make the robot oscillate, the specific gain value K' during oscillations is linearly correlated with the distance to the observed objects. The robot can then detect the oscillations to perceive distance. The scaling of optical flow with this strategy comes from a combination of the time delay in the control system with the fact that the effects of control actions do not vary with height (more details are provided in Supplementary Section 1). Here, we use the gain K' in a self-supervised learning (SSL) process in which the robot learns to associate the visual appearance of images to the gain K' (and hence the corresponding distance). After learning, oscillations are no longer necessary, allowing the robot to perceive distances immediately.

Landing

We first show the potential of the proposed self-supervised learning process by applying it to the task of optical flow landing. For landing, the relevant control action is the vertical thrust u, which is determined with a proportional gain control structure, $u = K\left(D^{*} - \hat{D}\right) + u_0$, where $D^{\star}$ is the desired constant divergence, defined here as $\left(\frac{v_z}{z}\right)^{*}$, $\hat{D}$ is the divergence estimated by means of computer vision methods and u_0 is the 'trim' thrust necessary for hovering flight. The flying robot, a Parrot Bebop 2.0 drone (shown in Fig. 2a), first performs three training landings. During training, the robot adaptively tunes its control gain so that it is continuously oscillating with the gain K proportional to the height z. Figure 2b shows data for a single training landing, with the gain over time as the blue line and the ground-truth height (determined by an external motion tracking system) as the red line. At the start of the landing, the robot starts increasing the optical flow control gain to achieve oscillations. Around $t=30$, the robot starts to oscillate, and it continuously reduces and increases the gain to land on the 'edge of oscillation'[27]. Each time step that the robot oscillates, it stores the control gain K and a texton histogram[31] $\mathbf{h}$ that represents the visual appearance of the landing surface.

After the training landings, we perform linear regression to find a function f that maps the texton histograms $\mathbf{h}_i$ in the training set to the corresponding control gains K_i. Subsequently, the regression function f is used during the test landings to estimate the gain K' purely from the visual appearance of the landing surface, that is, $\widehat{K'} = f(\mathbf{h})$. Figure 2c compares the estimate $\widehat{K'}$ (dotted blue line) with the height z (red line) obtained with the motion tracking system. The prediction is quite accurate, with a mean absolute error (MAE) of the estimate of $|\widehat{K' - K'^{*}}| = 0.24$, where $K'^{\star}$ is the appropriately scaled version of z. The immediate determination of the appropriate control gain $\alpha\widehat{K'}$ (with $\alpha \in (0,1)$, solid blue line in Fig. 2c) allows for high-performance optical flow landings, in this case, with $D^{\star} = -0.3$, without any oscillations at the end of the land-

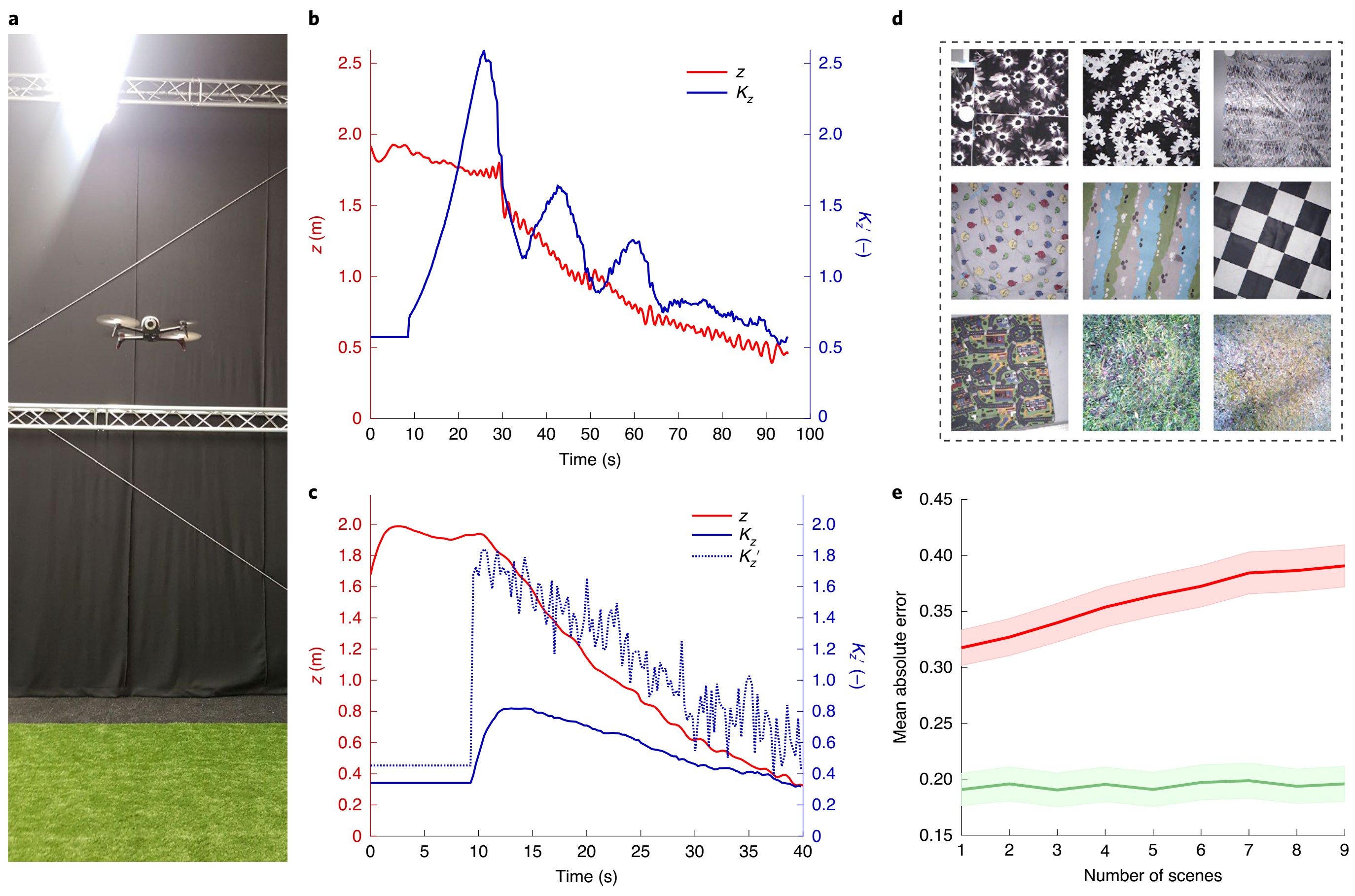


Fig. 2 | Landing results. a, Parrot Bebop 2 drone in the indoor flight arena. **b**, The adaptive gain and height during a 'training landing'. While oscillating, the drone learns to associate the visual appearance of the landing surface with the control gain K'. **c**, Height and the gain estimated from the visual appearance, $\widehat{K'}$, during a fast test landing with $D^* = -0.3$. The mean absolute error (MAE) of the estimate is $|\widehat{K'} - K'^*| = 0.24$, where K'^* is the appropriately scaled height z. **d**, Different landing surfaces, seven indoors and two outdoors. **e**, Effect of increasing the number of visually different landing surfaces on the MAE of the gain estimation. The red line shows the average performance for the straightforward, efficient linear learner used on board the robot. The green line shows the performance of a more complex learner, that is, a three-nearest-neighbour regressor based on computationally more costly computer vision features (a combination of Laws masks, edgeness histograms and hue histograms). It has a very good performance (MAE ≤ 0.20) even for the set of all nine environments. The shaded areas represent the standard error, that is, $\sigma/\sqrt{N}$, with the number of tests $N = 300$. Photo in **a** by G.C.H.E.d.C.

ing (cf. Supplementary Section 2). The proposed strategy outperforms state-of-the-art optical flow landing methods by allowing for quicker, high-performance optical flow landings that are not subject to oscillations close to the landing surface (Extended Data Fig. 1).

The proposed change from a pre-wired, purely optical-flow-based vision system to a learning-based system has implications for the performance specifications. In particular, these specifications should include how many landing surfaces a function f can learn and how well a learned function f will generalize to other landing surfaces. Insight into these matters aligns with the knowledge available in the field of machine learning[32,33], to which all major findings apply. For example, the number of visually different landing surfaces that can be learned with f depends on the complexity of the feature space and the learner. Figure 2d shows nine visually dissimilar scenes we have used to construct a dataset. Figure 2e shows that a simple learner, like the linear regressor on the robot (red line), can cope with fewer landing surfaces than more advanced computer vision algorithms, such as a three-nearest-neighbour regressor using more complex vision features (green line). Although the average MAE of the linear regressor increases from 0.32 to 0.38 when going from one to nine environments, the MAE of the nearest-neighbour regressor with more elaborate visual features stays at ~0.19. Furthermore, the generalization to other landing surfaces depends on the similarity between the training and test distributions. Having learned on a grass field, f will have a higher performance on a similar grass field than on a concrete road. However, as shown in Supplementary Section 5 and Supplementary Fig. 9, the robot can be aware of the change of environment. If it is uncertain about its estimates, it can retrigger the self-supervised learning procedures. In the worst case, the robot would be too confident about wrong gain estimates, leading to oscillations or to bad tracking of D^*. Both of these consequences can also be detected by the robot, which can retrigger the self-supervised learning.

Obstacle avoidance

We also apply the proposed self-supervised learning process to the task of obstacle avoidance. For this task, the robot will fly forward with a forward-looking camera, while continuously regulating the lateral flow ω to zero with a fixed control gain K: $u = K(\omega^* - \omega) + u_0$. Hence, in contrast to the landing task, the controlled motion axis is now orthogonal to the motion direction. The concept for initial obstacle avoidance during training is to fly forward until the point

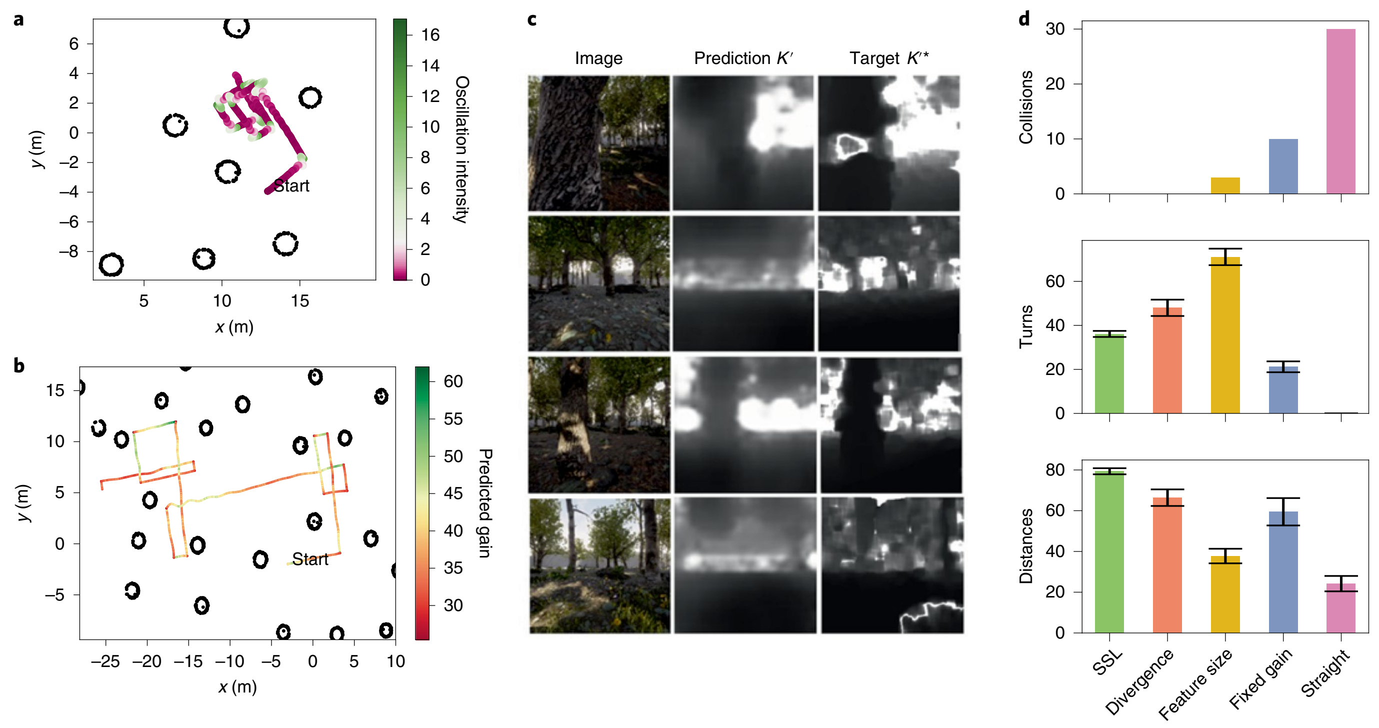


Fig. 3 | Simulated obstacle avoidance experiments. a, Trajectory of the robot during training. The agent flies forward, regulating horizontal flow with a fixed gain $K=20$ until it starts to oscillate. When oscillating, the drone can learn to map the perceived colour image to a target gain image K'^{*}. After oscillating for a second, the robot turns 90°. **b**, After having learned to map colour images to 'gain images', the robot can also decide to turn when its estimate of the average gain $\widehat{K'}$ is getting too low. This allows the robot to fly straight trajectories while flying faster. **c**, Images from the simulated forest environment, with corresponding gain predictions and targets (results on the test set). **d**, Comparison of different avoidance algorithms. 'SSL' (green) is the proposed method, 'Divergence' makes the robot turn when the optical flow divergence exceeds a threshold (orange), 'Feature Size' does not use optical flow but the size increase for divergence estimation (dark yellow), 'Fixed gain' is the initial training behaviour (blue) and 'Straight' is a baseline method in which the robot does not turn and collides with the first obstacle on its path (pink). The error bars show the standard error, that is, $\sigma/\sqrt{N}$, with number of runs $N=30$.

where the robot starts to oscillate. Because the robot uses the average flow ω for control, this will happen when the average distance to the objects in view is equal to or smaller than that for which K is the critical gain. The oscillation serves as an indication that the robot should stop and turn, and in the same time is used for learning to see dense distances by means of the visual appearance of the obstacles in view. After training, the robot can detect obstacles anywhere in its field of view by means of their visual appearance. This improves the obstacle detection and allows the robot to fly faster.

Figure 3 shows the results of dense distance estimation for a simulated robot (due to its simulated nature, referred to as 'agent') in a visually realistic forest environment created with the Unreal simulator[34] (Extended Data Fig. 3). Figure 3a shows the trajectory of the agent during training, flying at $25\,\mathrm{cm\,s^{-1}}$. When it oscillates, it uses the flow ω and gain K used by the controller to transform the dense optical flow to a target gain image K'^{*}, representing the distances (cf. Extended Data Fig. 2). Subsequently, it turns 90° to the left. After all data are gathered, we train a fully convolutional neural network (FCN) designed for monocular dense distance estimation[35] to map the colour images to the target gain images. Figure 3c shows example colour images, the predicted gain images that come out of the FCN, and the target gain images, all from the test set. After training, the FCN can predict dense distances, despite evident occasional artefacts in the target gain images. Figure 3b shows the agent's trajectory after learning, flying at $1\,\mathrm{m\,s^{-1}}$. The capability of seeing distances by means of visual appearance allows the agent to speed up without inducing collisions. Figure 3d contains a comparison with $N=30$ runs of the proposed SSL strategy (green bar) and a standard approach that turns when the divergence surpasses a threshold (orange bar), an approach that has been designed specifically to deal with frontal obstacles, using feature sizes instead of flow vectors[36] (dark yellow), the control strategy followed during learning, but executed at a higher speed (blue bar), and an approach that just flies straight (pink bar). The SSL strategy has no collisions, while achieving a greater distance than the initial fixed-gain strategy and the divergence-based approaches. The data shown in Fig. 3d correspond to a single threshold—that is, operating point—for each of the shown methods. Varying the thresholds per method leads to a 'distance-collision curve' per method, which also shows that the proposed SSL strategy has the best performance (Extended Data Fig. 4).

Finally, we show that the self-supervised learning strategy for obstacle avoidance also works on a flying robot (Fig. 4a). During training, the robot flies five times towards an obstacle at a forward velocity of $0.9\,\mathrm{m\,s^{-1}}$, while regulating the vertical optical flow to zero with the thrust. The robot stops when it oscillates. The trajectories of these flights are shown in Fig. 4b, while the Y position and the covariance of the optical flow and thrust (an indication of the amount of oscillation) are shown in Fig. 4c. The robot reliably stops at ~1 m from the obstacle. In this experiment, the drone continuously stores a texton histogram $\mathbf{h}$ and the covariance value c, which increases when going towards the obstacle (Supplementary Section 3 provides an extension of the proof of ref. [27] to a properly delayed control system). After the five training flights, a linear regression function f is learned that maps each $\mathbf{h}_i$ to c_i. During test flights, the robot uses $\hat{c}=f(\mathbf{h})$ and a corresponding threshold c_{T} to stop before the obstacle. After training, the robot can fly faster and stop at different distances from the obstacle, depending on c_{T}. Figure 4d shows

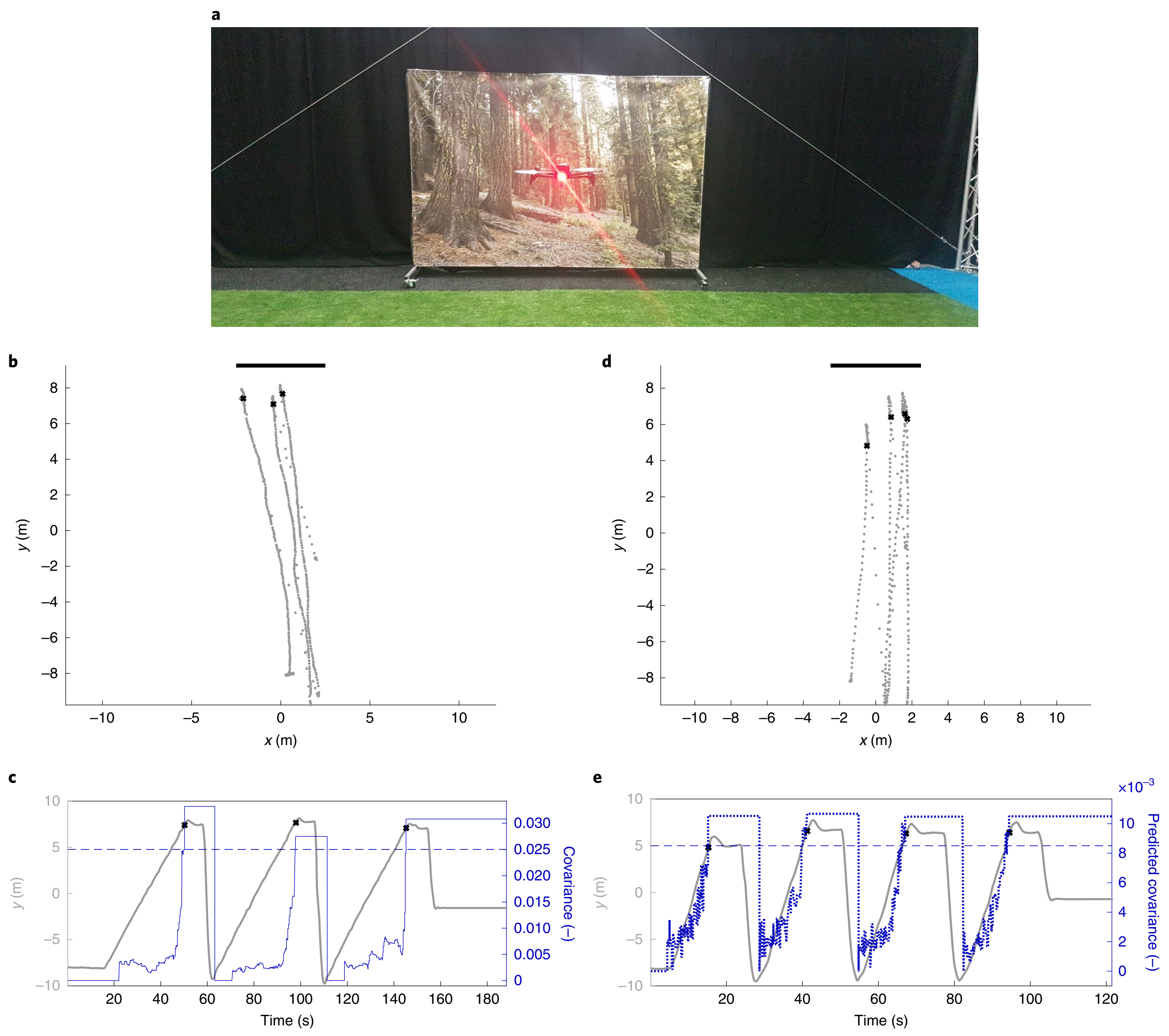


Fig. 4 | Real-world obstacle avoidance experiments. a, Parrot Bebop 2 drone flying towards a screen with a forest scene on it. **b**, Trajectories during training (grey dots). The drone attempts to keep the vertical flow to zero while flying forward at ~0.90 m s^{-1}. Close to the screen (black line), the drone starts to oscillate, leading to a large covariance of thrust and optical flow. The drone lands when the covariance surpasses a threshold (crosses). **c**, The *y* position of the drone (grey) and the amplitude of oscillations as measured by the covariance of thrust and flow (blue). The blue dashed line shows the covariance threshold used for stopping. While flying, the drone logs texton histograms representing the appearance and the covariance values for training. **d**, Drone trajectories (grey dots) after training. The drone maps texton histograms to covariance values and is able to stop (crosses) in front of the screen (black line), purely based on the appearance of the image, without oscillating. This allows the drone to fly faster (2.5–2.7 m s^{-1} here) and stop closer to or further from the screen if desired. **e**, The *y* position (grey) and covariance (blue) as predicted by the visual appearance over time. A lower threshold is used to stop slightly earlier and take into account the higher speed of the drone. Photo in **a** by G.C.H.E.d.C.

the trajectories for the robot flying forward with ~2.6 m s^{-1}, that is, almost three times as fast as during training, and with a threshold of $c_T = 0.0085$ (Fig. 4e).

Discussion

Robotics. The proposed self-supervised learning process resolves the major fundamental problems of optical flow by allowing the learning of complementary information from the visual appearance, which is promising for the field of robotics. For landing, the robot learned the appearance to set the control gains to achieve high-performance optical flow landing. In comparison with commonly used sonar or laser altimeters, vision-based landing is not limited in altitude. In addition, vision allows the extraction of richer information on the landing site, such as the locations of obstacles. Furthermore, the proposed approach generalizes beyond landing. The presented solution to the optical-flow control-gain tuning problem equally applies to heavily studied monocular algorithms for image-based visual servoing (IBVS)[37], visual odometry[38] and monocular simultaneous localization and mapping (SLAM)[39], which face the same issues for determining control gains.

For obstacle avoidance, the robot is able to learn dense distance estimates, which considerably improve the robot's navigation performance. Recently, in the field of computer vision, multiple self-supervised learning methods have been introduced to learn dense distance estimation from monocular video streams[40,41]. These methods involve deep neural networks that learn to estimate distances in a manner that minimizes the reprojection error based on projective geometry. The main difference with the proposed method is that these deep neural networks give unscaled distance estimates; that is, they still require extra sensors to map the 'distances' to a convenient unit like metres. The interesting property of the learning process proposed in this Article is that it provides a distance map in a unit that is directly relevant to the robot's control: the gain K that would make the robot oscillate if it were used for control. This is arguably even more useful than a scale in metres, which still requires the human designer to map it to the appropriate control gains. The proposed self-supervised learning approach also raises a number of questions that are relevant for successful application to autonomous robots. For example, is it always possible to learn distance estimation from visual appearance? Most natural environments provide rich distance information in visual appearance. Indeed, humans exploit a great variety of such pictorial depth cues, including aerial perspective, interposition (occlusion), relative height, object size, texture gradient and linear perspective[42,43]. Although depth perception is typically studied from a human-centred, ground-based viewpoint, most of these cues are also available when looking down at the ground (for example, think of houses and trees getting smaller and smaller when taking off in a plane, or the texture of grass and earth patches getting finer and finer). In the case that the environment would for some reason contain only little such information, the robot can still fall back on its basic optical flow control skills. Other learning questions, such as those pertaining to the number of environments that can be learned and the generalization to unseen environments, have been briefly touched on in this Article. Such questions align with major questions in the field of machine learning. However, having a self-supervised learning process on an autonomous robot adds an extra dimension, because the robot can influence the dataset by means of its actions. We hope that our study will further motivate research on how to best embed self-supervised learning processes in autonomous robots.

Biology. The developed learning process also has substantial implications for our view on flying-insect intelligence. Although the specifics and neural implementations of insect behaviours are surely different from those we have proposed for our flying robots, the main elements of the proposed self-supervised learning process are biologically plausible and provide a new point of view regarding the well-known empirical findings for flying insects. The plausibility is supported by three main observations. First, the system relies on oscillations to gauge distances in the environment. Oscillations are common in flying insects, such as the oscillating, hovering-like behaviour during learning of nest sceneries in bees[44]. Here, distances are reportedly estimated using parallax information, and a complete three-dimensional image of the hive is composed. The proposed theory suggests that, instead of exploiting parallax, the insects may use a stability-based distance estimation, which is less sensitive to wind disturbances[27] than the use of parallax (Supplementary Section 1c). Second, the proposed mechanism focuses on the learning of visual appearance (colour, texture and patterns). For insects, this could be the recognition of a flower species and its shape. It is well known that flying insects such as honeybees can learn complex patterns when faced with foraging tasks[45]. We have intentionally employed low-complexity vision methods for the robotic experiments to show that even limited visual learning capabilities can suffice to achieve increased behavioural success. More complex visual machinery, like the deep neural network used in the simulations, allows for better performance in terms of learning more complex objects and more different types of environment. Third, and most importantly, the core of the presented learning process is that it allows for increasingly successful behaviour. Learning processes are present in many species of insects. Behavioural improvements take place over the course of each forager's lifetime[46]. Honeybees are also able to learn complex visual patterns related to food sources[47]. In extension to these learning mechanisms, here we suggest that insects may learn visual patterns to improve their control skills. Hence, the effect of a newly presented cue set (for example, type of flower) should dominate over improvement by lifetime experience. Such effects have actually been observed in navigation experiments. For example, honeybees that were trained to feed close to a cylinder perform increasingly smooth and direct flight paths, substantially reducing flight duration[48]—an effect that was attributed to reinforcement learning. Our proposed, self-supervised learning process presents an alternative explanation for this effect, which, due to the lower sample complexity of supervised learning, also fits better with the low number of flights necessary for behavioural improvement. Whereas appearance learning has traditionally been associated with reward learning and navigation, we suggest that it is also important for control.

In general, the proposed learning process solves the major problems of optical flow, while its elements seem in line with the behaviours, sensors and (neuronal) capabilities of flying insects. This suggests that learning might play a far more important role in control behaviours, which are generally assumed to be hardwired. Therefore, ontogenic studies are called for rather than studies of 'trained' behaviours when analysing the impressive visual control skills of insects.

Methods

Landing experiments. A Parrot Bebop 2.0 drone was used for the experiments. Instead of the standard Parrot firmware, custom software from the Paparazzi UAV open-source autopilot was uploaded to the drone for the experiments. The experiment starts with learning a 'texton dictionary' by flying over the landing zone at different heights. The dictionary consists of 20 textons, that is, 20 prototypical 6×6-pixel image patches. Learning starts with a dictionary with all grey textons (R,G,B) = (128,128,128) and proceeds as follows. For every image, M image patches are extracted at random locations. Each image patch is matched to all textons, and assigned to the one with the smallest sum of squared differences (SSD). This texton is then moved towards the newly assigned image patch with the following learning rule: $\mathbf{t} \leftarrow \beta\mathbf{t} + (1-\beta)\mathbf{p}$, where $\mathbf{t}$ is the texton and $\mathbf{p}$ the image patch, both in vector representation. Training the dictionary only takes a few minutes, during which the parameter β is gradually reduced from 0.1 to 0.01, and then 0 to end the training. For the multi-environment experiment (discussed separately), the dictionary is learned on the training set. After training the dictionary, three landings 'on the edge of oscillation' are performed, meaning that an adaptive gain control is used that varies the K in the optical flow control loop $u = K(D^* - D)$, so that $c = \mathrm{cov}(u, D)$ is controlled to be equal to c^*: $K(t+T) \leftarrow K(t) + K(t)\left(P(c^* - c(t)) + I\int_0^t (c^* - c(t'))\mathrm{d}t'\right)$, where T is the sampling time step. The drone is first commanded in Paparazzi's 'NAV' mode (completely controlled using the external motion tracking system, consisting of 10 Optitrack Prime 17W motion capture cameras) to hover at 2 m above the landing surface. The autopilot is then switched to 'Module' mode, in which the vertical axis is completely controlled by the optical flow divergence controller. Optical flow is determined as follows. We first detect FAST corners[49] in the previous image with an active vision method based on ACT-CORNER[50]. Specifically, 25 'agents' move over the image in an informed way (following edges, speeding up in textureless areas, and so on), and stop when they have found a FAST corner. The tracking of the corners from the previous to the current image is done with the Lucas–Kanade algorithm[51]. Finally, the 'size divergence method'[27] is used for estimating the divergence $\hat{D}$, which is used for control. Before starting a landing, the optical flow controller regulates the divergence to be 0 ($D^* = 0$), so that the control gain K can increase to the right level. To actually start the landing, the drone changes the divergence setpoint to $D^* = -0.05$. During the landing, the drone constitutes the training set by storing the texton histogram $\mathbf{h}(t)$ with the corresponding gain $K(t)$ when oscillating. As in ref. [27], oscillation detection is done by looking at the covariance between the control actions u and the estimated divergence $\hat{D}$, where a large covariance implies the occurrence of oscillations. After the drone has made three such landings, a 'learn' process is activated while not in flight, which performs a linear least squares (LLS) fit of the data: $H\mathbf{w} = \mathbf{k}$, where H consists of the histograms $\mathbf{h}$ with a row of 1s in the last column to provide a bias, $\mathbf{w}$ is the vector of learned weights, and $\mathbf{k}$ is the vector with all gain values K corresponding to the texton histograms. The LLS was implemented with a singular value decomposition (SVD) of the data matrix, with a prior on the weights so as to favour small weight values. The total learning procedure (from flights to learned weights) takes on the order of 5–10 min (Supplementary Video 2). After learning, the drone is commanded to take off and hover again in NAV mode at the requested height. Now, when we switch to Module mode for landing, the control gain K of the optical flow control loop is continuously set by $K(t) \leftarrow \rho K(t-\Delta t) + (1-\rho)\alpha \widehat{K'(t)}$, with $\alpha \widehat{K'(t)} = \alpha(\mathbf{w} \cdot \langle \mathbf{h}(t), 1 \rangle)$, the 'stable gain factor' $\alpha = 0.5$ and a low-pass filter constant $\rho = 0.95$. The stable gain factor transforms the estimated oscillation-inducing control gain to a value that leads to high-performance control but does not self-induce oscillations. When $K(t)$ goes below a threshold value, the drone commences a landing flare. For determining the performance of the predictor f, we compare the estimated gain $\widehat{K'}$ and the scaled height z. Scaling has been performed by taking the gains K_i in the training data and the corresponding z_i and then performing a linear least squares fit: $\mathbf{k} = \gamma \mathbf{z}$, where $\mathbf{k}$ is the vector containing all K_i and $\mathbf{z}$ is the vector containing all z_i. The scalar parameter γ depends on the drone. In the experiments performed with the Parrot Bebop 2.0, $\gamma = 1.01$. Concerning processing, most of the computation is spent on the corner detection, Lucas–Kanade optical flow tracking (of a maximum of 25 corners) and the extraction of the texton distribution (basing its histogram on the extraction of 250 image patches per image). These processes take on average 1 ms, 20 ms and 10 ms, respectively, when running on the native, on-board dual 2-GHz core ARM Cortex A9 processor of the Bebop 2. The total processing time per frame is then 31 ms, which means that the vision processes run on-board at ~32 frames per second. Learning of a dataset of around 500 samples takes ~50 ms with LLS. Prediction with the linear learner requires 41 floating point operations (FLOPs), which takes almost no processing time (<<1 ms).

Multi-environment experiment. The data for the experiment were gathered by having the Parrot Bebop 2.0 land on the edge of oscillation over nine visually different landing areas, seven in the indoor flight arena 'the Cyberzoo' at the Faculty of Aerospace Engineering at TU Delft and two outdoors over a grass landing surface. The test reported in the Article focuses on how many environments a learner can manage. An MAE for n environments is obtained as follows. We perform 300 tests, with each test selecting a random subset of n environments out of the total nine environments. For each environment, a contiguous 10% of the data are reserved as the testing data. The remaining 90% of data serve as training data. The 'simple learner' is a linear regressor as used in the robotic experiments, which uses a texton histogram with 30 textons in the dictionary. The 'complex learner' is a k-nearest-neighbour[32] regressor with $k = 3$ using a combination the following three feature types: hue, Laws mask[52] and edgeness histograms. The hue histograms are constructed by going over all pixels in the image (transformed to the hue saturation value colour space) and adding a pixel to a 10-bin hue histogram if the corresponding saturation and value of the pixel are larger than 0.2. We use convolutions of the nine Laws mask features with a grey-scale version of the image to create nine histograms. The limits of the 10 bins of the histograms are determined per Laws mask feature on the training set, so that on the theoretical 'average' image, each bin is expected to contain 10% of the samples. The edgeness histograms have 20 bins and capture per pixel the sum of absolute horizontal and vertical grey image gradients. That is, the edgeness is defined as $e = \frac{|I_x| + |I_y|}{2}$, where I_x and I_y represent the horizontal and vertical image gradient, respectively. The three types of histogram lead to a feature vector of $10 + 90 + 20 = 120$ features, as opposed to the 30 texton features for the simple learner.

Obstacle avoidance experiments in simulation. The simulator used is the visually very realistic Unreal simulator[53]. A forest environment has been created for the work in ref. [54], consisting of a central area with trees as the main obstacles. The terrain is slightly undulating and covered with grass, flowers, small bushes and occasional tree logs. Many elements, like branches and flowers, move with a varying wind. The outer border of the square central area is a solid, smooth grey wall. The simulated agent is a sphere, which moves forward at a fixed velocity, with 1 m s^{-1} as default. It has a forward-looking camera located at the centre of the sphere, from which 240 × 240-pixel images are captured by the UnrealCV package[34]. The Python code used to connect to the Unreal simulator, simulate the agent's dynamics and implement the control loops is included with this submission and is publicly available. In simulation, the agent has a fixed forward velocity and a constant height, controlling only the lateral dynamics. Hence, in contrast to the real-world experiments, in simulation u represents the sideways acceleration. There is no actuation noise. The camera inputs and vision processing take place at 10 frames per second, comparable with the vision processing of the front camera in real experiments. The delay of the vision measurements in simulation is five time steps (0.5 s). Each pair of incoming images is processed as follows with OpenCV[55] to obtain optical flow vectors: (1) maximally 200 FAST feature points are extracted from the image, (2) they are tracked from image t to image $t + 1$ by means of the Lucas–Kanade optical flow algorithm[51] and only successfully tracked features are kept, (3) the lateral flow is determined as the median of all flow vectors in the image, where in our simulation experiments we use the horizontal flow. When the optical flow control induces oscillations, detected by the covariance of the sideways acceleration command and the horizontal flow, a turn of 90° is executed, which is set to take 1 s. This way of controlling the agent is called 'fixed gain' control. After learning, the agent can also perform 'predictive gain' control, in which it uses the dense distance predictions to determine when to turn. To this end, it determines the average of the closest 50% of distance estimates in the vertically central region of the image (240 × 80 pixels). For the standard divergence controller that uses optical flow, the divergence is determined as follows. Per optical flow vector i, the divergence is determined as $D_i = \frac{\|q_i\|}{d_{\text{FoE}}}$, where $\|q_i\|$ is the magnitude of the flow vector projected onto the line connecting the initial point to the FoE, and d_{FoE} is the distance of the initial point to the FoE. Normally, a difficulty of determining the divergence this way is that it is already a challenge to determine the FoE, but in this simulation we can (correctly) assume the FoE to be in the centre of the image. The conditions for the divergence-based controller are rather benign; besides the known location of the FoE, the camera images from the simulator are not influenced by factors such as motion blur, rolling shutter effects, thermal noise, reflections and so on. This means that the tracking errors of the Lucas–Kanade optical flow algorithm are quite small compared to a real-world scenario. Still, the optical flow tracking process has some errors, so the optical flow variables used for control are subject to some noise. Moreover, the simulator contains slightly moving elements (like leaves and flowers) that violate the assumption of a static world. Finally, we have also compared the proposed SSL approach with a feature-size-based method. This method is based on ref. [36], in which it was argued that size increase is much more reliable for detecting frontal obstacles than optical-flow-based methods. Given two images, the method we implemented starts by extracting AKAZE features[56] from the images and matching them. Note that, in ref. [36], SURF features were used, but this method is no longer publicly available via OpenCV. If a pair of matching features passes the following tests, it is used for a single divergence determination, where (1) the second-best match is considerably worse than the first-best match and (2) the features are sufficiently similar in feature space. In the original article[36] there was an additional test ensuring that the feature size increases over time. However, this implies an a priori assumption that all divergences are positive. We have implemented this extra check and performed experiments with and without, but did not find any substantial difference, so we have left this out of the final algorithm. When a match is good, a template window is taken around the feature's position in the image at time t, with a size w determined as a factor $5\frac{1}{3}$ times the feature's size s. In the image at the following time step, $t + 1$, windows of different sizes are extracted with a constant factor m times w, with $m \in \{0.8, 0.9, 1.0, \ldots, 1.6\}$ (in ref. [36] it went from 1 to 1.5 with unknown step size), corresponding to a divergence range of $[-2.5, 3.75]$, taking into account a frames per second of 10. The discrete scaling steps lead to discrete divergence estimates. Our verification experiments in Supplementary Section 4 show that more accurate results are obtained if the scale is refined with a quadratic fit, reminiscent of the procedure to find sub-pixel disparities in stereo vision. The result is a distribution of divergences, of which we take the median in our experiments. Other percentiles of the distribution led to similar or worse results.

Determining dense distance maps. If a robot uses optical flow for control and it is experiencing self-induced oscillations, it can perform dense distance estimation, that is, for every pixel in its image. In a pinhole camera model, the relation of the translational optical flow between two static world points is proportional to the relation of their distances to the camera. When the robot oscillates, the flow used for control (in the case of the simulation the average flow ω) stands for the 'distance' K, which allows the determination of the distances for all optical flow values in the image: $\frac{\dot{x}_i}{\omega} = \frac{K}{K_i'^{*}}$, where $\dot{x}_i$ is the flow at a pixel i, and $K_i'^{*}$ the critical gain for that pixel. When oscillating, the robot can store a colour image together with a dense distance image K'^{*}, allowing it to perform supervised dense distance learning. In our simulator, we obtain dense optical flow by means of the Farnebäck optical flow algorithm[57].

The supervised dense distance learning involved a deep neural network inspired by ref. [35]. In particular, we used an FCN, with, as basis VGG-16, excluding the fully connected top three layers (that normally contain most of the parameters). The VGG-16 layers are kept fixed during training. We add three convolutional layers with 8, 8 and 16 channels, respectively. Each of these layers has a 3 × 3 filter size, ReLu activation functions and is followed by an up-sampling with factor 2. The final layer is a convolutional layer with sigmoid activations, outputting a single-channel 56 × 56-pixel depth map, matching the size of the dense gain maps produced with the help of active optical-flow control. The network performs a total of 29.5 million FLOPs and contains 14.75 million parameters (with a float64 representation, this leads to memory usage of 123.2 MB). To gather the training data, we performed 1,000 simulated flights of 120 simulation seconds each, flying at 0.25 m s^{-1}. This resulted in a total of 718 pairs of RGB and gain images, augmented by factor 2 by means of left/right flipping. Training involved 1,000 epochs consisting of 100 batch iterations with a batch size of 32 images on an NVidia GeForce GTX 1080 Ti, taking 29 h. The forward pass during inference took ~400 ms (2.5 frames per second). Hence, for use by less powerful embedded processors, it would be best to further optimize the network architecture and manner of compilation, preferably with the processing hardware in mind (as explored in refs. [58,59]).

Obstacle avoidance experiments in the real world. A Parrot Bebop 2.0 drone was used for the experiments, again replacing the standard firmware with the Paparazzi open-source autopilot. In these experiments the frontal camera is used. The experiments start again with learning a texton dictionary, similarly as for the landing experiments, but now varying the distance to the obstacles in front of the drone. Subsequently, five approaches to the obstacle are made with the optical flow control. For real-world obstacle avoidance, we let the drone fly forward at a constant velocity, in our experiments ~0.90 m s^{-1}. While flying forward, the drone keeps the vertical optical flow in the image constant by controlling its thrust: $u = K(\omega^{\star} - \omega)$, where K is fixed in this case. When the drone gets close to an obstacle, the gain will start to induce oscillations. Upon oscillation detection $c > c_T$, the drone stops. All the time when going forward, the drone logs the texton histogram $\mathbf{h}(t)$ and the covariance used for oscillation detection, $c(t)$. After the five training runs, a linear estimator is learned with LLS: $H\mathbf{w} = \mathbf{c}$, where H is the matrix of histograms expanded with a last column of 1s for the bias, $\mathbf{w}$ are the learned weights and $\mathbf{c}$ are the covariance values corresponding to the training histograms. As for the landing task, the LLS was performed with an SVD with a prior on small weight values. Also here, the total learning procedure (from flights to learned weights) is on the order of 5–10 min (Supplementary Video 2). After learning the drone can detect imminent collisions by means of the covariance value estimated from the texton histogram, $\hat{c}(t) = \mathbf{w} \cdot \langle \mathbf{h}(t), 1 \rangle$. The drone will stop when $\hat{c}(t) > c_T$. Although the front camera images (640 × 640 pixels) are larger than the bottom camera images (240 × 240 pixels), processing times are identical to the landing experiments. This is because both the corner detection and the texton method are sampling-based, and hence do not exhaustively process the image.

Statistical information. For the simulation results, we have determined the significance of the differences in the covered distances by different methods ($n = 30$ per method). Given the non-normal nature of the distributions, we employed a two-tailed, randomized, bootstrap method with 10,000 samples to compare the distributions. The proposed SSL approach covers a larger distance than the other methods, with $P = 0.004$, 0.000 and 0.006 in comparison with the divergence, feature-size and fixed-gain method, respectively. The divergence method is also significantly better than the feature-size method ($P = 0.000$), but is not significantly different from the fixed-gain method ($P = 0.383$). Also, the fixed-gain method reaches significantly larger distances than the feature-size method ($P = 0.007$). Similarly, we have performed two-tailed bootstrap tests with the results of the comparison between the simple and complex learner (Fig. 2e, $n = 300$). For all data points, that is, different numbers of scenes, the difference is statistically significant, with $P = 0.000$.

Data availability
All data are publicly available at https://doi.org/10.34894/KLKP1M.

Code availability
All code bases (in C, Python and MATLAB) that have been used in the experiments are publicly available at https://doi.org/10.34894/KLKP1M.

Received: 28 July 2020; Accepted: 2 December 2020;
Published online: 19 January 2021

References
1. Floreano, D. & Wood, R. J. Science, technology and the future of small autonomous drones. *Nature* **521**, 460–466 (2015).
2. Franceschini, N., Pichon, J.-M. & Blanes, C. From insect vision to robot vision. *Philos. Trans. R. Soc. Lond. B* **337**, 283–294 (1992).
3. Webb, B. Robots in invertebrate neuroscience. *Nature* **417**, 359–363 (2002).
4. Franceschini, N. Small brains, smart machines: from fly vision to robot vision and back again. *Proc. IEEE* **102**, 751–781 (2014).
5. Gibson, J. J. *The Ecological Approach to Visual Perception* (Houghton Mifflin, 1979).
6. Collett, T. S. Insect vision: controlling actions through optic flow. *Curr. Biol.* **12**, R615–R617 (2002).
7. Srinivasan, M. V., Zhang, S. W., Chahl, J. S., Stange, G. & Garratt, M. An overview of insect-inspired guidance for application in ground and airborne platforms. *Proc. Inst. Mech. Eng. G* **218**, 375–388 (2004).
8. Srinivasan, M. V., Zhang, S.-W., Chahl, J. S., Barth, E. & Venkatesh, S. How honeybees make grazing landings on flat surfaces. *Biol. Cybern.* **83**, 171–183 (2000).
9. Baird, E., Boeddeker, N., Ibbotson, M. R. & Srinivasan, M. V. A universal strategy for visually guided landing. *Proc. Natl Acad. Sci. USA* **110**, 18686–18691 (2013).
10. Ruffier, F. & Franceschini, N. Visually guided micro-aerial vehicle: automatic take off, terrain following, landing and wind reaction. In *Proc. 2004 IEEE International Conference on Robotics and Automation* Vol. 3, 2339–2346 (IEEE, 2004).
11. Herisse, B., Russotto, F. X., Hamel, T. & Mahony, R. Hovering flight and vertical landing control of a VTOL unmanned aerial vehicle using optical flow. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems* 801–806 (2008); https://doi.org/10.1109/IROS.2008.4650731
12. Alkowatly, M. T., Becerra, V. M. & Holderbaum, W. Bioinspired autonomous visual vertical control of a quadrotor unmanned aerial vehicle. *J. Guid. Control Dyn.* **38**, 249–262 (2015).
13. Van Breugel, F., Morgansen, K. & Dickinson, M. H. Monocular distance estimation from optic flow during active landing maneuvers. *Bioinspir. Biomim* **9**, 2 (2014).
14. Howard, D. & Kendoul, F. Towards evolved time to contact neurocontrollers for quadcopters. In *Proc. Australasian Conference on Artificial Life and Computational Intelligence* 336–347 (Springer, 2016).
15. Scheper, K. Y. W. & de Croon, G. C. H. E. Evolution of robust high speed optical-flow-based landing for autonomous MAVs. *Rob. Auton. Syst.* (2020); https://doi.org/10.1016/j.robot.2019.103380
16. Hagenaars, J. J., Paredes-Vallés, F., Bohté, S. M. & de Croon, G. C. H. E. Evolved neuromorphic control for high speed divergence-based landings of MAVs. Preprint at https://arxiv.org/pdf/2003.03118.pdf (2020).
17. Santer, R. D., Rind, F. C., Stafford, R. & Simmons, P. J. Role of an identified looming-sensitive neuron in triggering a flying locust's escape. *J. Neurophysiol.* **95**, 3391–3400 (2006).
18. Muijres, F. T., Elzinga, M. J., Melis, J. M. & Dickinson, M. H. Flies evade looming targets by executing rapid visually directed banked turns. *Science* **344**, 172–177 (2014).
19. Nelson, R. & Aloimonos, J. Obstacle avoidance using flow field divergence. *Pattern Anal. Mach.* **I**, 1102–1106 (1989).
20. Green, W. E. & Oh, P. Y. Optic-flow-based collision avoidance. *IEEE Robot. Autom. Mag.* **15**, 96–103 (2008).
21. Conroy, J., Gremillion, G., Ranganathan, B. & Humbert, J. S. Implementation of wide-field integration of optic flow for autonomous quadrotor navigation. *Auton. Robots* **27**, 189 (2009).
22. Zingg, S., Scaramuzza, D., Weiss, S. & Siegwart, R. MAV navigation through indoor corridors using optical flow. In *2010 IEEE International Conference on Robotics and Automation* 3361–3368 (IEEE, 2010).
23. Milde, M. B. et al. Obstacle avoidance and target acquisition for robot navigation using a mixed signal analog/digital neuromorphic processing system. *Front. Neurorobot.* **11**, 28 (2017).
24. Rind, F. C., Santer, R. D., Blanchard, J. M. & Verschure, P. F. M. J. in *Sensors and Sensing in Biology and Engineering* (eds. Barth, F. G. et al.) 237–250 (Springer, 2003).
25. Hyslop, A. M. & Humbert, J. S. Autonomous navigation in three-dimensional urban environments using wide-field integration of optic flow. *J. Guid. Control Dyn.* **33**, 147–159 (2010).
26. Serres, J. R. & Ruffier, F. Optic flow-based collision-free strategies: from insects to robots. *Arthropod Struct. Dev.* **46**, 703–717 (2017).
27. De Croon, G. C. H. E. Monocular distance estimation with optical flow maneuvers and efference copies: a stability-based strategy. *Bioinspir. Biomim.* **11**, 1–18 (2016).
28. Stevens, J.-L. & Mahony, R. Vision based forward sensitive reactive control for a quadrotor VTOL. In *Proc. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems* (*IROS*) 5232–5238 (IEEE, 2018).
29. Sanket, N. J., Singh, C. D., Ganguly, K., Fermüller, C. & Aloimonos, Y. GapFlyt: active vision based minimalist structure-less gap detection for quadrotor flight. *IEEE Robot. Autom. Lett.* **3**, 2799–2806 (2018).
30. Bertrand, O. J. N., Lindemann, J. P. & Egelhaaf, M. A bio-inspired collision avoidance model based on spatial information derived from motion detectors leads to common routes. *PLoS Comput. Biol.* **11**, e1004339 (2015).
31. Varma, M. & Zisserman, A. Texture classification: are filter banks necessary? In *Proc. 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* Vol. 2, II–691 (IEEE, 2003).
32. Mitchell, T. et al. Machine learning. *Annu. Rev. Comput. Sci* **4**, 417–433 (1990).
33. Bishop, C. M. *Pattern Recognition and Machine Learning* (Springer, 2006).
34. Qiu, W. et al. UnrealCV: virtual worlds for computer vision.In *Proc. 25th ACM International Conference on Multimedia* 1221–1224 (ACM, 2017); https://doi.org/10.1145/3123266.3129396
35. Mancini, M., Costante, G., Valigi, P. & Ciarfuglia, T. A. Fast robust monocular depth estimation for obstacle detection with fully convolutional networks. In *Proc. 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems* (*IROS*) 4296–4303 (IEEE, 2016).
36. Mori, T. & Scherer, S. First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles. In *Proc. IEEE International Conference on Robotics and Automation* 1750–1757 (IEEE, 2013); https://doi.org/10.1109/ICRA.2013.6630807
37. Chaumette, F., Hutchinson, S. & Corke, P. in *Springer Handbook of Robotics* (eds. Siciliano, B. & Khatib, O.) 841–866 (Springer, 2016).
38. Scaramuzza, D. & Fraundorfer, F. Visual odometry [tutorial]. *IEEE Robot. Autom. Mag.* **18**, 80–92 (2011).
39. Engel, J., Schöps, T. & Cremers, D. LSD-SLAM: large-scale direct monocular SLAM. In *Proc. European Conference on Computer Vision* (*ECCV*) 834–849 (Springer, 2014).

40. Zhou, T., Brown, M., Snavely, N. & Lowe, D. G. Unsupervised learning of depth and ego-motion from video. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* 1851–1858 (IEEE, 2017).
41. Gordon, A., Li, H., Jonschkowski, R. & Angelova, A. Depth from videos in the wild: unsupervised monocular depth learning from unknown cameras. Preprint at https://arxiv.org/pdf/1904.04998.pdf (2019).
42. Gibson, J. J. *The Perception of the Visual World* (Houghton Mifflin, 1950).
43. Brenner, E. & Smeets, J. B. J. Depth perception. *Stevens' Handb. Exp. Psychol. Cogn. Neurosci.* **2**, 1–30 (2018).
44. Lehrer, M. & Bianco, G. The turn-back-and-look behaviour: bee versus robot. *Biol. Cybern.* **83**, 211–229 (2000).
45. Stach, S., Benard, J. & Giurfa, M. Local-feature assembling in visual pattern recognition and generalization in honeybees. *Nature* **429**, 758–761 (2004).
46. Andel, D. & Wehner, R. Path integration in desert ants, *Cataglyphis*: how to make a homing ant run away from home. *Proc. R. Soc. Lond. B* **271**, 1485–1489 (2004).
47. Dyer, A. G., Neumeyer, C. & Chittka, L. Honeybee (*Apis mellifera*) vision can discriminate between and recognise images of human faces. *J. Exp. Biol.* **208**, 4709–4714 (2005).
48. Fry, S. N. & Wehner, R. Look and turn: landmark-based goal navigation in honey bees. *J. Exp. Biol.* **208**, 3945–3955 (2005).
49. Rosten, E., Porter, R. & Drummond, T. Faster and better: a machine learning approach to corner detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **32**, 105–119 (2010).
50. de Croon, G. C. H. E. & Nolfi, S. ACT-CORNER: active corner finding for optic flow determination. In *Proc. IEEE International Conference on Robotics and Automation* (*ICRA 2013*) (IEEE, 2013); https://doi.org/10.1109/ICRA.2013.6631243
51. Lucas, B. D. & Kanade, T. An iterative image registration technique with an application to stereo vision. In *Proc. International Joint Conference on Artificial Intelligence* Vol. 81, 674–679 (ACM, 1981).
52. Laws, K. I. *Textured Image Segmentation*. PhD thesis, Univ. Southern California (1980).
53. Games, E. *Unreal Simulator* (Epic Games, 2020); https://www.unrealengine.com
54. Kisantal, M. *Deep Reinforcement Learning for Goal-directed Visual Navigation* (2018); http://resolver.tudelft.nl/uuid:07bc64ba-42e3-4aa7-ba9b-ac0ac4e0e7a1
55. Pulli, K., Baksheev, A., Kornyakov, K. & Eruhimov, V. Real-time computer vision with OpenCV. *Commun. ACM* **55**, 61–69 (2012).
56. Alcantarilla, P. F. & Solutions, T. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *IEEE Trans. Patt. Anal. Mach. Intell.* **34**, 1281–1298 (2011).
57. Farnebäck, G. Two-frame motion estimation based on polynomial expansion. In *Proc. 13th Scandinavian Conference on Image Analysis* 363–370 (ACM, 2003).
58. Sanket, N. J., Singh, C. D., Fermüller, C. & Aloimonos, Y. PRGFlow: benchmarking SWAP-aware unified deep visual inertial odometry. Preprint at https://arxiv.org/pdf/2006.06753.pdf (2020).
59. Wofk, D., Ma, F., Yang, T.-J., Karaman, S. & Sze, V. Fastdepth: fast monocular depth estimation on embedded systems. In *Proc. 2019 International Conference on Robotics and Automation* (*ICRA*) 6101–6108 (ICRA, 2019).
60. Herissé, B., Hamel, T., Mahony, R. & Russotto, F.-X. The landing problem of a VTOL unmanned aerial vehicle on a moving platform using optical flow. In *Proc. 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* 1600–1605 (2010); https://doi.org/10.1109/IROS.2010.5652633
61. Ho, H. W., de Croon, G. C. H. E., van Kampen, E., Chu, Q. P. & Mulder, M. Adaptive gain control strategy for constant optical flow divergence landing. *IEEE Trans. Robot.* (2018); https://doi.org/10.1109/TRO.2018.2817418

Acknowledgements
We thank M. Kisantal for creating the UnrealCV environment that was used in this work. We also thank F. Muijres, M. Karasek and M. Wisse for their valuable feedback on earlier versions of the manuscript.

Author contributions
All authors contributed to the conception of the project and were involved in the analysis of the results and revising and editing the manuscript. G.C.H.E.d.C. programmed the majority of the software, with help from C.D.W. for the implementation in Paparazzi for the real-world experiments. G.C.H.E.d.C. performed all simulation and real-world experiments. G.C.H.E.d.C. created all graphics for the figures.

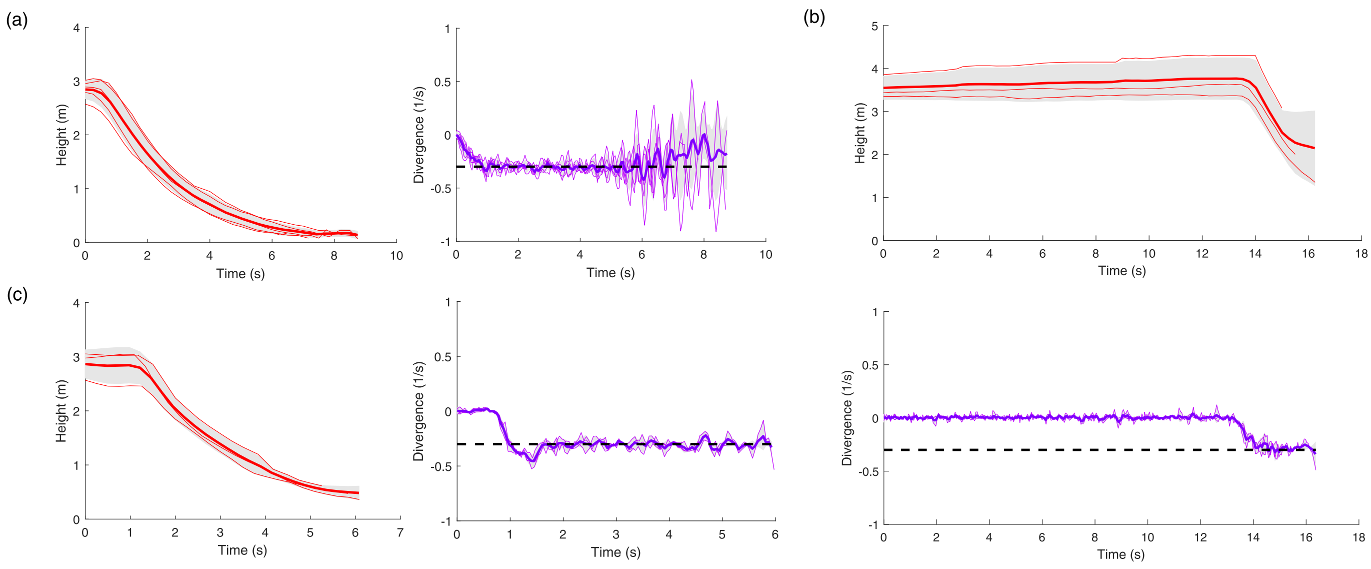
Competing interests
The authors declare no competing interests.

Additional information
Extended data is available for this paper at https://doi.org/10.1038/s42256-020-00279-7.

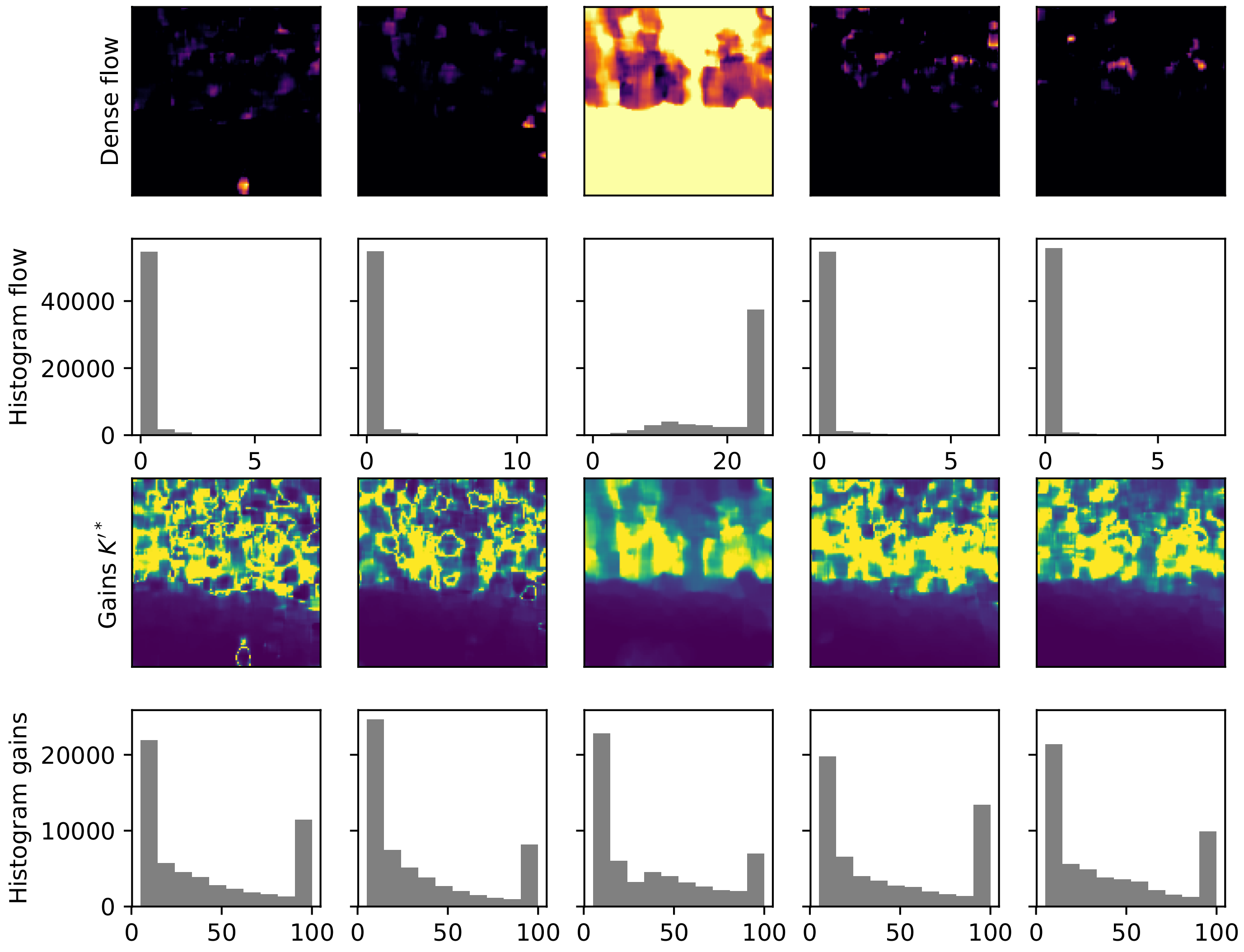
Supplementary information is available for this paper at https://doi.org/10.1038/s42256-020-00279-7.

Correspondence and requests for materials should be addressed to G.C.H.E.d.C.

Peer review information *Nature Machine Intelligence* thanks Yiannis Aloimonos and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.

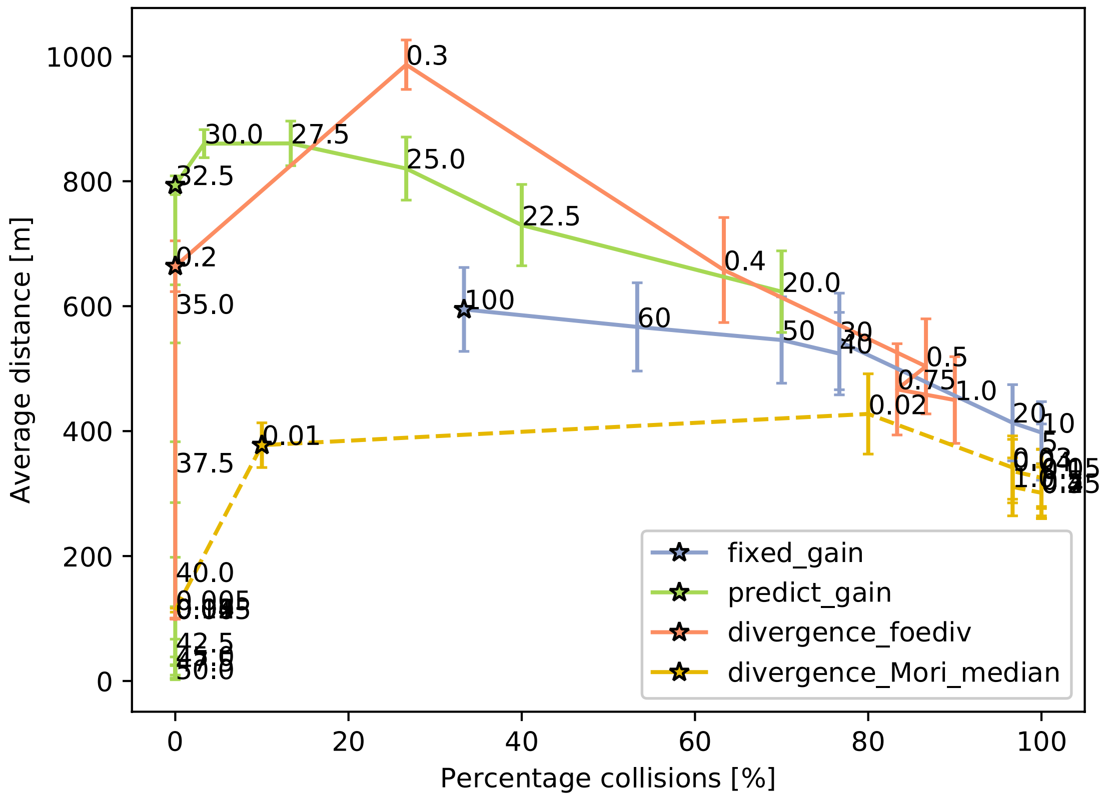


Extended Data Fig. 1 | Landing performance for different landing strategies. For each strategy we show the height (red lines), and the divergence (purple lines) over time. Thick solid lines show the average over 5 landings, while the grey area shows the sampled standard deviation. All landings aimed for a divergence of D*=-0.3 and started in hover, that is, at D=0. **a**, Landing with a fixed gain K (as in, for example,[60]), resulting in oscillations close to the surface. **b**, Landing with an adaptive gain as in[61]. The performance during landing is good, but slowly increasing the gain takes a rather long time (in the order of 14 seconds). **c**, Landing with the proposed self-supervised learning strategy. It can immediately start landing and has a good performance all the way down. Note the quicker landings with respect to a fixed gain and the absence of evident oscillations in the divergence.



Extended Data Fig. 2 | Dense distance estimation from optical flow and oscillations. *First (top) row:* Dense horizontal optical flow images, determined with the Farnebäck algorithm[57]. *Second row:* Histogram of each optical flow image. The x-axis represents the optical flow in pixels/frame. *Third row:* Corresponding dense gain (distance) images. *Fourth row:* Histogram of each gain image. Although the optical flow changes substantially during oscillatory motion of the flying robot (rows 1–2), the gain values are of comparable magnitude (rows 3–4). Smaller optical flow does lead to noisier gain estimates.

Extended Data Fig. 3 | Example images from the Unreal simulator. These images are taken at 960 × 960 pixel resolution. The environment contains a variety of trees and lighting conditions. The border of the environment is a smooth, grey, stone wall (bottom right screenshot). The flowers on the ground and the leaves of the trees move in the wind. The simulated flying robot perceives the environment at a 240 × 240 pixel resolution.



Extended Data Fig. 4 | Distance-collision curves for the four different avoidance methods. The curves are obtained by varying the parameters of the methods that balance false positives (turning when it is not necessary) and false negatives (not detecting an obstacle). For the predictive gain method this is the gain threshold, for the divergence methods that use optical flow vectors or feature size increases it is the divergence threshold, and for the fixed gain method it is the control gain. The numbers next to the graphs are threshold / parameter values. Per parameter value, N=30 runs have been performed, and the error bars show the standard error $\sigma/\sqrt{N}$. The stars indicate the operating points shown in Fig. 3 in the main article.