Evolution of robust high speed optical-flow-based landing for autonomous MAVs

Scheper, Kirk Y.W.; de Croon, Guido

DOI
10.1016/j.robot.2019.103380

Publication date
2020

Document Version
Accepted author manuscript

Published in
Robotics and Autonomous Systems

This is an accepted manuscript preprint of published work available at https://doi.org/10.1016/j.robot.2019.103380. Please cite with Kirk Y.W Scheper and Guido C.H.E. de Croon, Evolution of robust high speed optical-flow-based landing for autonomous MAVs, Robotics and Autonomous Systems 124, 2020.

Evolution of Robust High Speed Optical-Flow-Based Landing for Autonomous MAVs

Kirk Y.W. Scheper*, Guido C.H.E. de Croon
Micro Air Vehicle Laboratory, Aerospace Engineering, Delft University of Technology, Delft, Netherlands

HIGHLIGHTS

- Neurocontrollers for high speed optical-flow-based landing were automatically developed with better results than the user-defined baseline.
- The optimized behavior was seamlessly transferred to the real world with the aid of a closed loop control system.
- Preprocessing the sensory input allowed the behavior to be tested with both a CMOS camera and Dynamic Vision Sensor with no noticeable changes to the performance.
- No significant performance differences were observed despite clear differences in the neurocontroller input.

ARTICLE INFO

Article history:
Initially Received 31 March 2019
Revised 15 October 2019
Accepted 15 November 2019

Keywords:
Evolutionary Robotics
Bio-Inspired Landing
Reality Gap
High Speed Flight

ABSTRACT

Automatic optimization of robotic behavior has been the long-standing goal of Evolutionary Robotics. Allowing the problem at hand to be solved by automation often leads to novel approaches and new insights. A common problem encountered with this approach is that when this optimization occurs in a simulated environment, the optimized policies are subject to the reality gap when implemented in the real world. This often results in sub-optimal behavior, if it works at all. This paper investigates the automatic optimization of neurocontrollers to perform quick but safe landing maneuvers for a quadrotor micro air vehicle using the divergence of the optical flow field of a downward looking camera. The optimized policies showed that a piece-wise linear control scheme is more effective than the simple linear scheme commonly used, something not yet considered by human designers. Additionally, we show the utility in using abstraction on the input and output of the controller as a tool to improve the robustness of the optimized policies to the reality gap by testing our policies optimized in simulation on real world vehicles. We tested the neurocontrollers using two different methods to generate and process the visual input, one using a conventional CMOS camera and one a dynamic vision sensor, both of which perform significantly differently than the simulated sensor. The use of the abstracted input resulted in near seamless transfer to the real world with the controllers showing high robustness to a clear reality gap.

1. Introduction

Insects are a significant source of inspiration for developing novel solutions to autonomous flight of Micro Air Vehicles (MAVs). Even with limited computational and energy resources, insects are able to effectively complete challenging tasks with relatively complex behaviors. One behavior that is of particular interest, is landing.

Insects have been shown to primarily rely on visual inputs when landing, whereby many insects regulate a constant rate of expansion (or *divergence*) of optical flow [1] to perform a smooth landing [2]. This approach has inspired some robotic implementations of constant divergence landing strategies [3]. Although simple in concept, this approach is quite difficult to implement in reality. A direct implementation causes the robot to become unstable as the vehicle nears the ground. This is a result of the non-linear interaction between the vehicle control and sensing, in the presence of delay, measurement noise and environmental disturbance [4]. Different augmentations to the standard control scheme have been made, most, simply perform slow landings [3], use long landing legs to delay the onset of this instability and touch the ground before they occur or by switching to alternative measures like time-to-contact [5, 6] or velocity in-plane of the camera [7, 8]. More recently, there have been attempts to identify the instability and adapt the landing strategy by adjusting control gains [9, 10].

An alternative to these manually designed approaches is to have an optimization technique automatically develop a suitable solution that is robust to these instabilities. This optimization may reveal new solutions to this problem, and perhaps even alternative hypotheses on what flying insects such as honeybees may be doing. Some attempts have been made to do this [11] but to the best knowledge of the authors, none of these have been implemented on real world robots.

This is due in part, to the effects of the differences between the simulated environment, commonly used for behavioral optimization, and the real world. This resultant difference in the robotic behavior is commonly referred to as the *reality gap* [12, 13, 14]. Several approaches have been used to make controllers more robust to the reality gap with the most significant being adding appropriate and varied noise [15], co-evaluation of controllers in the real world to test transferability [16] and inspired by conventional control theory, there are approaches utilizing abstracted outputs from the neurocontroller with a closed loop control system to actively reduce the effect of reality gap [17, 18].

In this paper, we describe a method to optimize a quick

*Corresponding author
Email addresses: k.y.w.scheper@tudelft.nl (K.Y.W. Scheper); g.c.h.e.Croon@tudelft.nl (G.C.H.E. de Croon)
ORCID(s): 0000-0003-2770-5556 (K.Y.W. Scheper); 0000-0001-8265-1496 (G.C.H.E. de Croon)

but safe landing maneuver for a quadrotor MAV equipped with a downward facing camera. The neurocontrollers are given only the divergence of the optical flow field from the camera as input and its time derivative. Although this abstracted input may reduce the possible behaviors the controllers can express, building on the work in [19], we will demonstrate that this abstraction leads to a robust transfer from simulation to reality after virtual optimization.

The following consists of a summary definition of optical flow in Section 2. The flight platform and simulation environment is then described in Section 3. Next, the performance of conventional constant divergence landing approaches are presented in Section 4 to provide some baseline performance to compare the optimized policies against. The evolutionary setup and neural models used for the neurocontrollers is then described in Section 5. This is followed by a presentation and analysis of the optimized policies in Section 6. The reality gap and the results from the real world experiments are presented in Section 7 and Section 8 from which we draw some conclusions in Section 9.

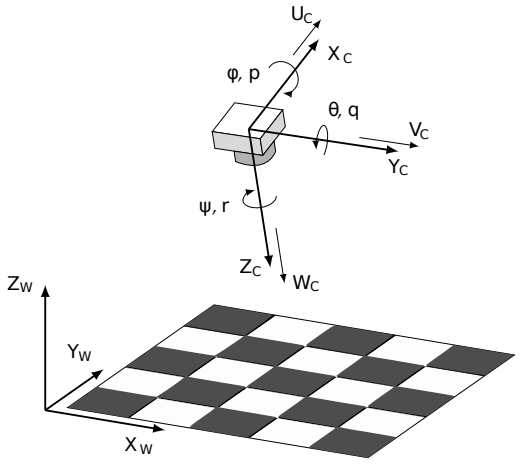


Figure 1: Definitions of the world ($\mathcal{W}$) and camera ($\mathcal{C}$) references frames. Also shown are the Euler angles (ϕ, θ, ψ), rotational rates (p, q r), and translational velocities (U, V, W) that describe the motion of $\mathcal{C}$ [22].

2. Optical Flow Definition

To perform optical flow based landing as in [20, 21, 9, 10], we must first define the optical flow parameters. The formulation here is a summarized version of that presented in [20]. This algorithm provides a good trade-off of accurate optical flow estimates while using relatively limited computational resources. This allows the perception and control loop to operate at high frequency and low latency on the embedded flight platform used in this paper. Alternative optical flow estimation methods could be used given that the estimation runs fast enough to facilitate the flight control. An investigation of the accuracy and reliability of the optical flow estimation is out of the scope of this paper.

If we assume that we have a downward-looking camera overlooking a static planar scene, as shown in Fig. 1, we can derive the perceived optical flow as the result of the camera ego-motion. The derivation of this optical flow model relies on the two reference frames, the inertial world frame is denoted by $\mathcal{W}$ and the camera frame centered at the focal point of the camera denoted by $\mathcal{C}$. In each of these frames, position is defined through the coordinates (X, Y, Z), with (U, V, W) as the corresponding velocity components. The orientation of $\mathcal{C}$ with respect to $\mathcal{W}$ is described by the Euler angles ϕ, θ, and ψ, denoting roll, pitch, and yaw, respectively. Similarly, p, q, and r denote the corresponding rotational rates.

The camera ego motion can be related to the optical flow, and visual observables based on the pinhole camera model [23] with camera pixel coordinates are denoted by (x, y), while (u,v) represent optical flow components, measured in pixels per second. These can be non-dimensionalized using the intrinsic calibration of the camera.

$$\begin{aligned} u &= -\frac{U_C}{Z_C} + \frac{W_C}{Z_C}x - q + ry + pxy - qx^2 \\ v &= -\frac{V_C}{Z_C} + \frac{W_C}{Z_C}y + p - rx - qxy + py^2 \end{aligned} \tag{1}$$

Eq. (1) shows that the optical flow of a point can be resolved into translational and rotational components. Since the latter is independent of the three-dimensional structure of the visual scene, these expressions can be derotated if information on the rotational rates of the camera is available. This derotation leads to pure translational optical flow components, denoted by (u_T, v_T). Moreover, if the scene is a planar surface, the depth Z_C of all visible world points are interrelated through:

$$Z_C = Z_0 + Z_X X_C + Z_Y Y_C \tag{2}$$

where Z_0 is defined as the distance to the surface along the optical axis of the camera, and Z_X and Z_Y represent the slopes of the planar scene with respect to the X- and Y-axis of $\mathcal{C}$.

In [23], the relation between the position of an arbitrary point in $\mathcal{C}$ and its projection onto the image plane is given by $(x, y) = (X_C/Z_C, Y_C/Z_C)$. Consequently, Eq. (2) may also be written in the form:

$$\frac{Z_C - Z_0}{Z_C} = Z_X x + Z_Y y \tag{3}$$

Further, let the *scaled velocities* of the camera ϑ_x, ϑ_y, and ϑ_z be defined as follows:

$$\vartheta_x = \frac{U_C}{Z_0}, \quad \vartheta_y = \frac{V_C}{Z_0}, \quad \vartheta_z = \frac{W_C}{Z_0} \tag{4}$$

Then, according to the derivations in [20], substituting Eq. (3) and Eq. (4) into Eq. (1) leads to the following expressions for translational optical flow:

$$\begin{aligned} u_T &= (-\vartheta_x + \vartheta_z x)(1 - Z_X x - Z_Y y) \\ v_T &= (-\vartheta_y + \vartheta_z y)(1 - Z_X x - Z_Y y) \end{aligned} \tag{5}$$

From Eq. (5), and under the aforementioned assumptions, the scaled velocities, which provide non-metric information

on camera ego-motion, can be derived from the translational optical flow of multiple image points. ϑ_x and ϑ_y are the opposites of the so-called *ventral flows*, a quantification of the average flows in the X- and Y-axis of C respectively [9]. Hence, $\omega_x = -\vartheta_x$ and $\omega_y = -\vartheta_y$. On the other hand, ϑ_z is proportional to the *divergence* of the optical flow field, $D = 2\vartheta_z$ [9].

The flow divergence can alternatively be estimated simply by the relative change in the distance (l) between any two points at over time (t) [10]. This method is referred to as *size divergence* (D_{s_t}). A reliable estimate of the divergence ($\hat{D}$) can be generated by averaging the divergence estimate from a set of N points in the image.

$$\begin{aligned} D_{s_t} &= \frac{1}{\Delta t}\frac{l_{t-\Delta t} - l_t}{l_{t-\Delta t}} \\ \hat{D} &= \frac{1}{N}\sum_{i=1}^{N} D_{s_t} \end{aligned} \tag{6}$$

In this work, we used a FAST corner detector and a Lukas–Kanade tracker implemented using OpenCV as in [10], we refer the reader there for more details. Throughout this paper, N is limited to 100 if there were more than 100 points or simply all points if fewer have been tracked.

3. Flight Platform and Simulation Environment

The flight platform used in this paper is the Parrot Bebop 2 quadrotor MAV[1], a picture of this vehicle flying in our indoor test environment can be found in Fig. 2. This vehicle is equipped with a 780 MHz dual-core Arm Cortex A9 processor, forward and downward facing CMOS cameras, sonar, and barometer enabling autonomous flight capabilities for up to 25 minutes. Full 3D flight control is enabled with the onboard use of the open source PaparazziUAV autopilot software [24]. In this work, we extract global optical flow from the downward facing camera using the Lucas–Kanade optical flow method executed onboard the vehicle [25][2].

Throughout the paper, an Optitrack[3] motion capture system was used to measure a ground truth of the vehicle position and motion. As the control task in this paper is only in the vertical axis, this ground truth measure was communicated to the vehicle to facilitate the control of the lateral axis of the vehicle. This was however not used in the vertical loop where instead the optical flow estimated onboard the vehicle was used.

To optimize our behavior in simulation, we first need a model of the vehicle. To this end we use a simple dynamical model of the vehicle, which is restricted to vertical motion only. The thrust generated by the rotors is modeled as a first order response with the dynamics defined in (7).

$$(\Delta t + \tau_T)\dot{T}_i = T_{sp} - T_{i-1} \tag{7}$$

[1] https://www.parrot.com/nl/en/drones/parrot-bebop-2

[2] Code used in this paper can be found https://github.com/kirkscheper/paparazzi/tree/updated_event_based_flow

[3] https://optitrack.com/

Figure 2: Parrot Bebop 2 quadrotor MAV equipped with SEEM1 Dynamic Vision Sensor.

where T_{sp} is the thrust set-point and spin-up spin-down time constant τ_T has a nominal value of 0.02 s. The thrust output is limited in the range [-0.8·g, 0.5·g] as a conservative model the maximum acceleration of the real vehicle.

The model used to describe the divergence estimation is based on the work presented in [21]. The observed divergence is the result of adding latency to the true divergence along with two types of noise, simple white noise drawn from $N(0,\sigma_w^2)$ and an additional noise proportional to the divergence magnitude drawn from $N(0,\sigma_p^2)$. [21] identified typical values for σ_w and σ_p as 0.1 s^{-1} and the latency L in the range of [50, 100] ms. We use similar nominal values for the standard deviations σ_w and σ_p are 0.1 s^{-1} and an exaggerated range for the latency L in the range of [1,4] samples or [20, 133] ms. Some additional computational jitter has been included, this simulates the situation of missed frames which happens when there are either insufficient or too many image features to be tracked. The chance of a missed frame is randomly determined with a given probability held constant for each simulation run randomly drawn from the range [0, 0.2].

4. Baseline Performance

Before we start to optimize a divergence based landing controller, let us first investigate the naive approach of a constant gain, constant divergence landing as studied in [3, 26, 4].

Fig. 3 shows the time history of a two constant divergence landing controllers performing a landing with a divergence set-point of 0.5. Controller C_1 has a relatively high gain and C_2 a low gain. The controllers are activated 1 s after the simulation starts. Fig. 17 shows the steady-state response of the controller with the time history control signal super-imposed.

These plots show that controller C_1 quickly reaches the desired divergence but as the vehicle descends the controller

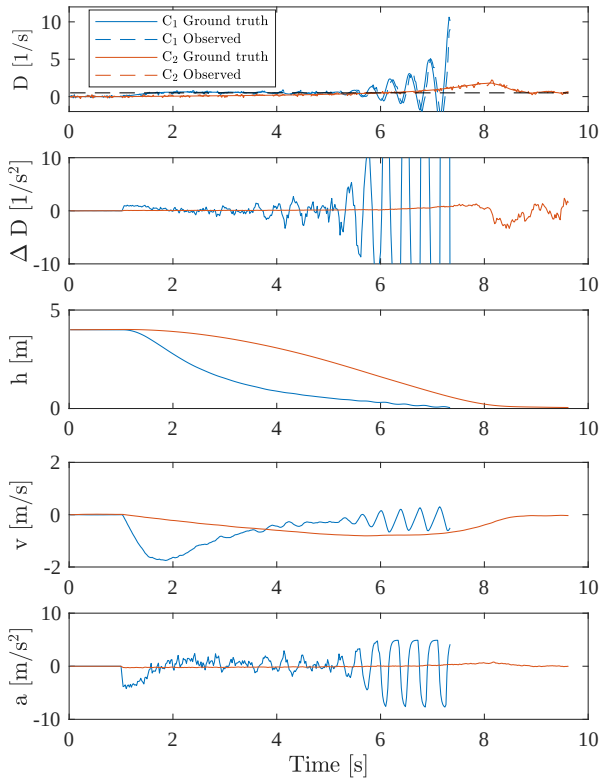


Figure 3: Time history of a constant divergence landing with two controllers of differing gain. C_1 is a controller with high gain and C_2 with low gain.

becomes increasingly unstable. This instability is due to delays in the sensing and control loop and is well described in [4]. In contrast, C_2 does not quickly achieve the desired divergence due to the low gain in the controller. Additionally, due to the non-linear relationship between the divergence and acceleration, the vehicle overshoots the desired set-point before finally slowing. Despite this poor tracking performance, the low gain does delay the onset of instability observed with C_1, resulting in a rather smooth landing profile.

5. Evolutionary Optimization

Every evolutionary process is defined by: a population of candidate individuals, each with a given policy which must be evaluated; a way to evaluate these policies; a selection mechanism to filter out bad policies; and a method to alter the individuals to generate new policies. Here, we use a mutation-only evolutionary algorithm similar to $(\mu + \lambda)$ approach, where a population is maintained from which offspring are generated using a mutation operator. Offspring that are better than members of the population replace these members. The difference with the standard implementation is that we retest the current population on a random set of simulation parameters with every generation, rather than once as is commonly done. With the high level of non-determinism in our simulated environment, this ensures no individuals are preferred simply because they were tested on an *easy* set of conditions.

Table 1
Evolutionary parameters

Name	Value
Number of Generations	250
Number of Runs	4
Range of delays	[1, 4] samples
Range of computational jitter probability	[0, 0.2]
Range of divergence noise (σ_w^2)	[0.05, 0.15] 1/s
Range of divergence noise (σ_p^2)	[0, 0.25] 1/s
Range of thrust time constant (τT)	[0.005, 0.04]
Range of simulation frequency	[30, 50] Hz

The evaluation of the individuals is done by simulating the policy on four independent simulation runs, initializing the vehicle at a standstill from four different altitudes, namely 2, 4, 6 and 8 m. The simulation ends when the vehicle exceeds 15 m above the ground, gets within 5 cm of the ground or exceeds 30 s of simulation time.

We use a multi-objective approach here, where the individuals must minimize three fitness functions measured at the end of a simulation: the total time to land (f_1); the final height (f_2); and the final velocity (f_3). NSGA-II is used to perform a non-dominated sorting of the population and determine which individuals are better than others in this multi-objective framework [27].

All the simulation parameters mentioned in Section 3 are randomly perturbed and set at the start of a generation. The ranges of the evolutionary parameters used in this work are summarized in Table 1. Altogether, this should encourage the optimization to develop a policy that can reliably land quickly and safely from different altitudes. This is all developed using the DEAP framework [28] with multi-threaded implementation utilizing scoop in Python[45].

The policy of each individual is encoded in a simple neural network. The neural potential (γ) of each neuron in the neurocontroller is updated with a simple discrete time Euler integration as described in Eq. (8).

$$\gamma(t) = \gamma(t-1) + \dot{\gamma}(t-1)\Delta t \tag{8}$$

where t represents the current time step and Δt is the time step of the integration.

The input to the neurocontroller is the simulated divergence and the derivative of the divergence $\Delta D = \frac{D_t - D_{t-\Delta T}}{\Delta t}$. The output was used to control the thrust of the vehicle leading to an acceleration. We utilized three types of neural networks to investigate the effect of recurrent connections on the evolved solution. We implemented a feed-forward neural network (hereafter referred simply as NN), a recurrent neural network (RNN) and a continuous time recurrent neural network (CTRNN). All networks had three layers, the first with 2 neurons, the hidden layer with 8 neurons and the output with 1 neuron.

[4]The software is openly available at: https://github.com/DEAP/deap

[5]Software used for the evolutionary process in this paper is openly available at: https://github.com/kirkscheper/divergence_landing

5.1. NN

The neural potential of an NN is defined by the instantaneous inputs to the network such that the potential of a neuron i in a given layer l connected to N^{l-1} neurons in the previous layer is determined simply by:

$$\gamma_i^l = \sigma^l(\sum_{j=1}^{N^{l-1}} w_{ij}^l \gamma_j^{l-1}) + \theta_i^l + I_i^l \tag{9}$$

where w_{ij} is the weight of the neural connection between the neuron i in layer $l = (2\pm 3\pm \cdots)$ and a given neuron j is the previous layer, θ is the neural bias, I is the external input to the neuron and σ is the activation function of the neuron. Here, the activation function is linear for the output layer and the Rectified Linear Unit (ReLU) function for the hidden layer. We can rewrite this equation using Eq. (8) to generate the neural dynamics.

$$\Delta t\dot{\gamma}_i = -\gamma_i + \sigma(\sum_{j=1}^{N} w_{ij}\gamma_j) + \theta_i + I_i \tag{10}$$

5.2. RNN

The NN has no effective way of explicitly considering previous states in determining it's current action. The behavior is simply a result of the emergent interaction of the vehicle actions and the environment. Adding explicit memory may enable the vehicle to exhibit more complex behaviors. RNNs are not only affected by an external input and the weighted sum of connected neurons but also by a weighted internal connection to the previous potential as shown in Eq. (11).

$$\dot{\gamma}_i = \gamma_i r_i + \sigma(\sum_{j=1}^{N} w_{ij}\gamma_j) + \theta_i + I_i \tag{11}$$

where r is the weight applied to the recurrent connection. Like the NN, the activation function is linear for the output layer and the ReLU function for the hidden layer.

Again, using Eq. (8), we can derive the neural dynamics:

$$\Delta t\dot{\gamma}_i = \gamma_i(r_i - 1) + \sigma(\sum_{j=1}^{N} w_{ij}\gamma_j) + \theta_i + I_i \tag{12}$$

5.3. CTRNN

One pitfall of the RNN is that the recurrent connection does not consider the time between updates. This can be an important consideration for systems with variable time steps between updates. As such, we have also implemented the classic CTRNN as shown in Eq. (13) [29].

$$(\Delta t + \tau_i)\dot{\gamma}_i = -\gamma_i + \sum_{j=1}^{N} w_{ij}\sigma(\gamma_j + \theta_j) + I_i \tag{13}$$

where τ is its time constant ($\tau > 0$). The hyperbolic tangent activation function is used here ($\sigma(x) = \tanh = (e^x - e^{-x})/(e^x + e^{-x})$).

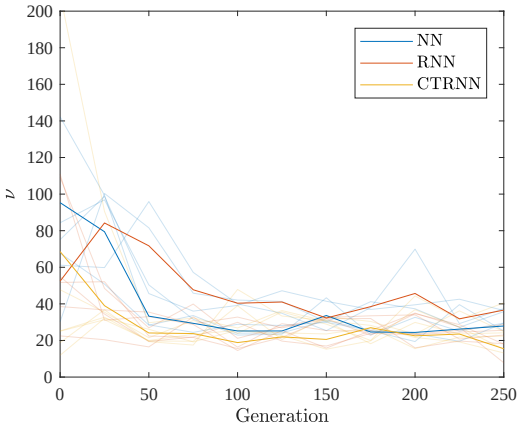


Figure 4: Performance of genetic optimization measured with ν which is the ratio of the encapsulated volume and area of the pareto front. A smaller ν would suggest a general improvement in the minimization of all individuals while they spread out over the available optimum policies. The pareto front was generated by evaluating the entire population at each generation to the same set of simulated environmental conditions.

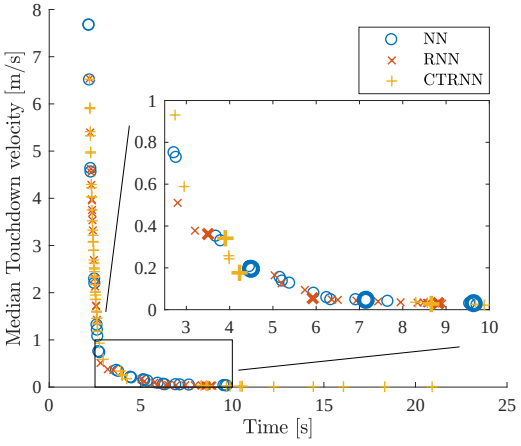


Figure 5: Performance of pareto front tested on 250 evaluations. Individuals selected for further analysis are bold.

6. Evolution Results

The use of the non-dominated sorting and selection, results in the genetic optimization spreading the population of policies over the pareto front of the fitness landscape. As such, to evaluate the performance of the optimization, it is sensible to look at the ratio of the encapsulated volume and the area of the pareto front ($\nu = volume/area$) as shown in Fig. 4. This figure shows that the performance generally improves before leveling off after about 150 generations. This performance is also mostly stable as the performance remains flat after 150 generations. This also shows that this trend is consistent over the multiple initializations of the optimization and over the different types of neural architecture.

Fig. 5 shows the accumulated Pareto front of all individuals from the different runs of the optimization. This figure only shows the performance on the touchdown velocity and the time fitness as the fitness based on the height was consis-

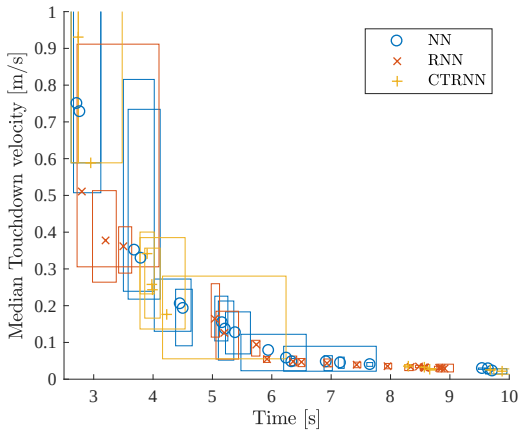


Figure 6: Sensitivity analysis of a portion of pareto front showing 25^{th}, 50^{th} and 75^{th} percentile of performance over 250 evaluations.

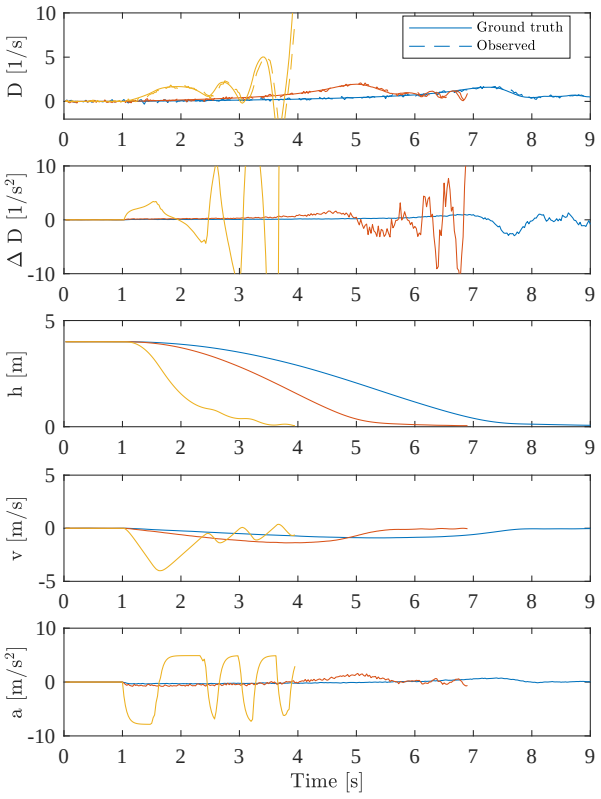


Figure 7: Vehicle states and observations from simulation with NN_1 (blue), NN_2 (red) and NN_3 (yellow).

tently minimized for all individuals. As such, all optimized policies converged to perform the desired landing task. This figure shows how the policies are spread over the inherent trade-off of reliable quick vs soft landing. The performance seems not to be strongly correlated with the neural architecture as all three types are represented in the pareto front. Additionally, the three architectures seem well distributed.

To investigate the sensitivity of the performance to different environmental conditions, we subjected the individuals of the pareto front to a validation test with 250 simulations while varying the environmental settings of the run. All individuals where subjected to the same set of conditions to make for a fair comparison. Fig. 6 shows that the individuals that optimized to have slow and soft landings have a small spread in the fitness performance whilst individuals that were optimized to faster landings have a larger variation in landing speed. This suggests that the policies that perform faster landings may show oscillatory behavior causing their touchdown speed to vary depending on the sensor noise or environmental perturbations. We will investigate this more below.

As it is not feasible to analyse them all, three individuals from the each architecture are be selected for further analysis of the landing behavior. The remainder of this section will dive deeper into the types of behaviors optimized by the different neurocontrollers.

6.1. NN

Three neurocontrollers were selected from the Pareto front for some further analysis, their performance is shown in Fig. 7 and their steady-state response in Fig. 18. Controller NN_1 is a slow lander, NN_3 is a fast lander and NN_2 is intermediate.

Looking first at the world states in Fig. 7, we can see that NN_1 and NN_2 perform smooth landings with little oscillation and touchdown at very low velocities. Examining the steady-state response, NN_1 is similar to the low gain baseline C_2 except that instead of being a constant gain for all inputs, NN_1 is piece-wise linear with a high gain for positive values D and a low gain for negative values. This asymmetric control scheme is a significant result as this will delay the onset of the oscillation seen in higher gain controllers.

NN_2 is similar to NN_1 but has a higher gain and a noticeable gradient in the ΔD with reduced thrust at negative ΔD. This is also an interesting result as a negative ΔD occurs when the vehicle is slowing while descending or accelerating while ascending. In both cases, it would indeed be desirable to reduce the control input.

NN_3 has clear oscillations and is similar to the high gain controller C_1 but is even higher gain, this causes the controller to act as a *bang-bang* controller. This type of control will cause the rotors of the vehicle to try to spin up and down very often, however, due to their inertia, they do not achieve the desired values. As such this control scheme relies on the simulated spin-up and spin-down reaction time of the rotors (τ_T) for the desired behavior to work well and will likely not transfer well to the real world. This may also explain why the controllers on the left of the pareto front in Fig. 6 have vertically elongated bounding boxes.

6.2. RNN

The selected RNN neurocontrollers are shown in Fig. 8 and Fig. 19. RNN_1 and RNN_2 show similar behavior to the policies with NN_1 and NN_2. These relatively high gain landings with a gradient on the ΔD seems to be a reliable way to perform this type of high speed yet smooth landing. RNN_3 is a little different than NN_3 in that high speed landings with a negative divergence rate have a lower throttle response. This would occur when the vehicle is descending quickly

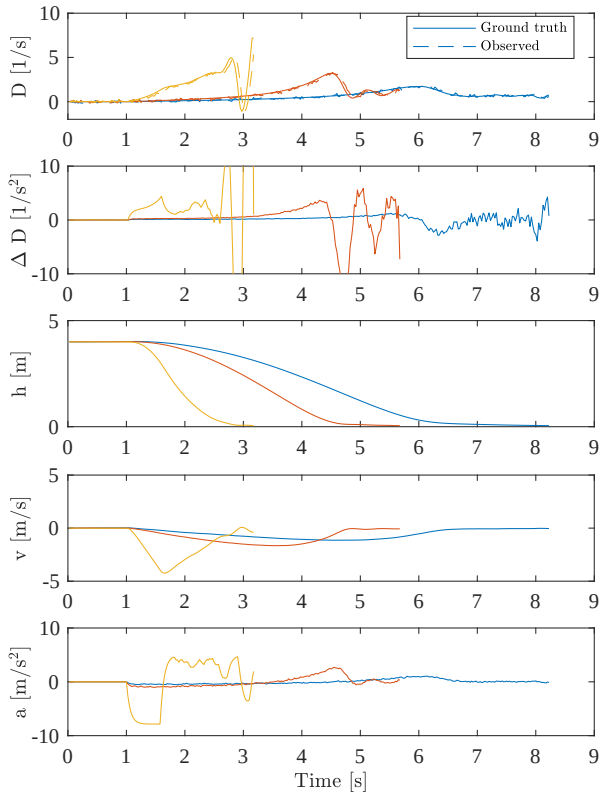


Figure 8: Vehicle states and observations from simulation with RNN_1 (blue), RNN_2 (red) and RNN_3 (yellow).

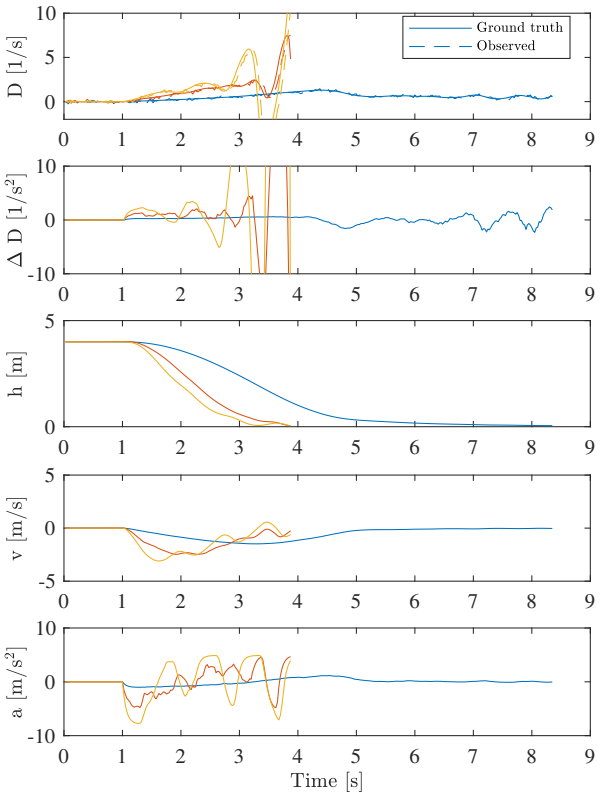


Figure 9: Vehicle states and observations from simulation with $CTRNN_1$ (blue), $CTRNN_2$ (red) and $CTRNN_3$ (yellow).

but slowing down. This control scheme would ensure that the vehicle doesn't over react when the vehicle is descending quickly but not accelerating towards the ground. This results in a reduced oscillatory behavior with the RNN_3 controller.

6.3. CTRNN

The three CTRNN neurocontrollers selected from the Pareto front in Fig. 9 and Fig. 20 all show variations on a similar control scheme. Effectively, these controllers split the control scheme into 4 segments, depending on the sign combination of the inputs D and ΔD. When descending with an increasing divergence, the vehicle will try to decelerate. When descending with an decreasing divergence, the vehicle will decelerate less aggressively. When ascending with increasing rate, the vehicle will reduce thrust. Finally, when the vehicle is descending with a negative divergence rate, the vehicle will reduce thrust less aggressively than the increasing rate case. This results in high speed yet smooth landings with little oscillation. This approach is similar to that shown by RNN_3. Portions of these controllers seem discretized, but as they only show the steady state throttle response, the temporal response may be more smooth.

7. Reality Gap

Everything discussed so far has been in a simulated environment, one that is a significant simplification of reality to facilitate high speed evaluation of the neurocontrollers. As we move to the real world, we can therefore expect various differences leading to a reality gap. This section aims to identify some of these differences.

Let us first look at the control of the vehicle, in simulation, the output of the neurocontroller was acceleration, which after being fed through a low pass filter to simulate the spin-up of the rotors was implemented by the simulated rotors. In reality, this desired acceleration must first be converted to a thrust command for the rotors. If we use a naive approach here, we can simply determine a linear transform from desired acceleration to thrust, the results of which are shown in the top plot of Fig. 10. This figure shows a set of real world neurocontroller landings using a constant scaling factor for desired acceleration to thrust. As the vehicle starts to move the command tracking is good but as it starts to descend the tracking degrades. This is almost certainly due, in part, to the unmodeled drag and non-linear aerodynamic effects of descending through the downwash of the propellers.

This poor tracking leads to a noticeable reality gap in the landing performance. This can be reduced with the use of a closed loop controller as proposed in [18], instead of a linear transform. The results using a Proportional-Integral (PI) controller, minimizing the error between the commanded and measured acceleration on the vehicle, are shown in the bottom plot of Fig. 10. The controller effectively abstracts away from the raw motor commands to a desired acceleration, which significantly improves the tracking performance allowing us to cross this reality gap.

The goal of this paper is to highlight the use of abstracted

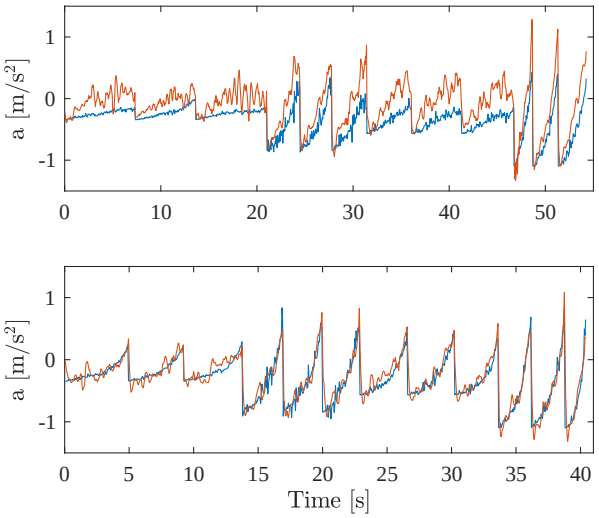


Figure 10: Thrust command (blue) and subsequent vehicle acceleration (red) for some real world landings. (Top) Unmodeled dynamics such as drag and other non-linear aerodynamic effects lead to a poor thrust command tracking using a naive approach. (Bottom) The tracking error can be substantially reduced with the use of a simple closed loop PI controller.

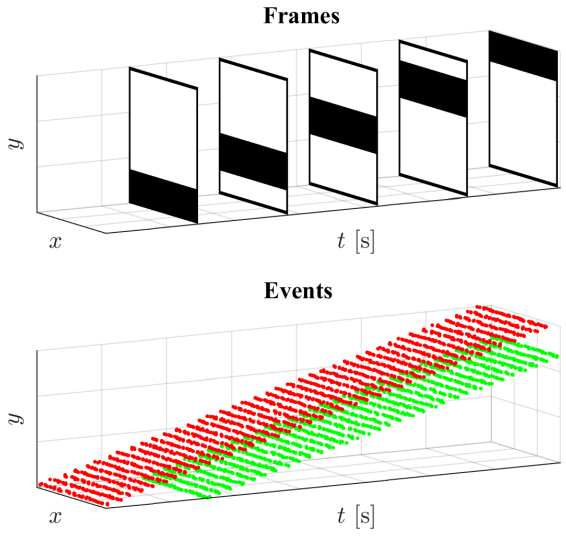


Figure 11: Depiction of the difference between frame-based and event-based vision data. Frame-based cameras measure light at all the pixels synchronously resulting in an image frame, optic flow is generated by identifying a feature in the current image in a previous image, the translation is the flow. Event-based cameras asynchronously generate an event for each pixel with a microsecond timestamp, optic flow is generated by fitting a plane through the 3D surface (x,y,t) of the most recent events.

inputs to improve the robustness of optimized policies to differences in the input. To do this we will investigate the performance of the vehicle with the use of two different types of cameras with significantly different divergence signal output performance characteristics.

The first camera uses the monochromatic information from the bottom looking CMOS camera built into the Parrot Bebop 2 using the size divergence estimation method described in [10]. The second camera is the Insightness SEEM1 Dynamic Vision Sensor (DVS) using the efficient plane fitting optical flow estimation technique described in [9]. This event based camera does not generate frames as a conventional image sensor but rather measures logarithmic light changes at each pixel independently and asynchronously. This makes the sensor conditioned to operate with high speed motion with low latency and relatively low data throughput. A simple comparison of the difference in data from the frame-based and event-based data can be seen in Fig. 11. This type of camera has been previously used to facilitate high speed landings as shown in [9]. A schematic showing the optical flow processing pipeline for the CMOS and event-camera can be found in Fig. 12.

Table 2
Camera properties

Property	Simulated	CMOS	DVS
Sensor	-	mt9v117	SEES1
Resolution used	-	240 x 240	262 x 262
Field of View	-	58°	79°
Divergence Rate	[30, 50] Hz	45 Hz	100 Hz

Fig. 13 shows a comparison of the divergence estimation error from these two cameras highlighting how different the output of these camera is and how different they both are to the camera statistics used in simulation. Some additional differences in the camera properties are summarized in Table 2. We will investigate in the next chapter if these differences result in a significant reality gap.

8. Flight Test Results

Using the closed loop PI controller to control the thrust as described in the previous section, we performed a set of landings with some of the neurocontrollers identified in Section 5. All flights were initiated from a steady hover at an altitude of 4 m. Fig. 14 shows the results from the controllers NN_1 and NN_2, Fig. 15 shows the results from RNN_1 and RNN_2 and Fig. 16 shows the results from $CTRNN_1$ and $CTRNN_2$. These results are plotted for both the CMOS and DVS cameras to see how well these two sensors affect the landing profile. The results from simulation have also been plotted to see how well the real world performance fits with the simulated, a measure for the eventual reality gap. NN_3, RNN_3 and $CTRNN_3$ were not tested in reality as their relatively high touchdown velocity in simulation may cause damage to our real world vehicle. This is in-fact a benefit of the multi-objective optimization scheme used here, the user can simply choose a policy that performs the trade-off of the fitness functions as desired.

In spite of the differences in the generation of the divergence between the two cameras, the landing performance is very similar. These two systems are also so similar to the simulated landing that the plot is hardly visible. This would suggest that the eventual reality gap is small despite significant differences in the way the input was generated.

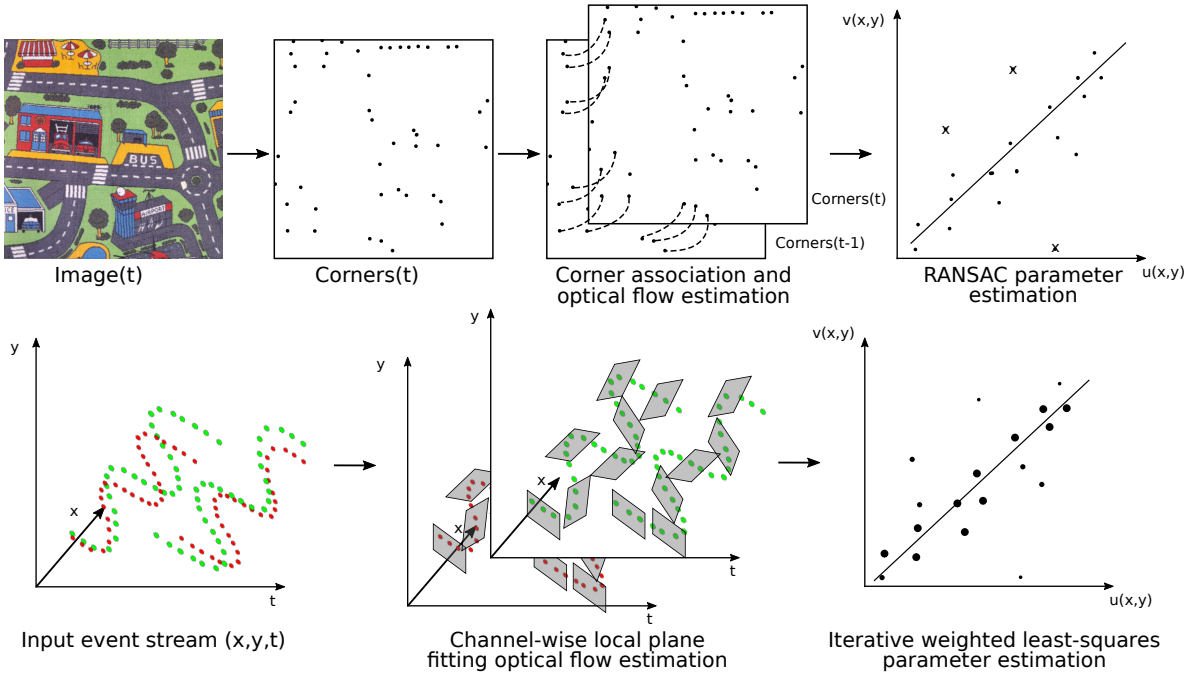


Figure 12: Process schematic of the optical flow computation for the CMOS camera (top) and the event-camera (below).

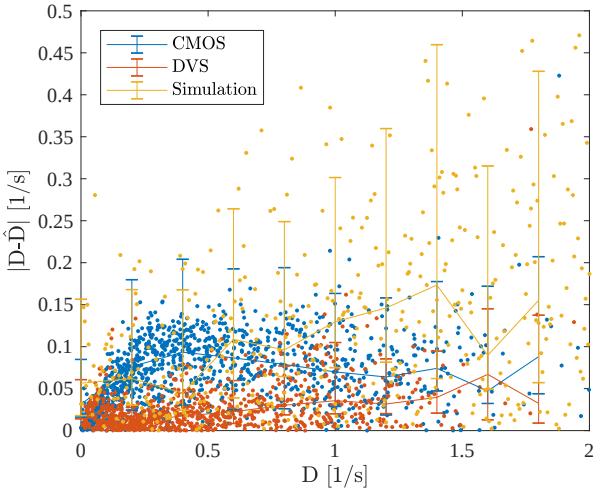


Figure 13: Divergence estimation error for the CMOS (blue) and DVS (red) cameras. The trend and distribution of these two cameras are quite different.

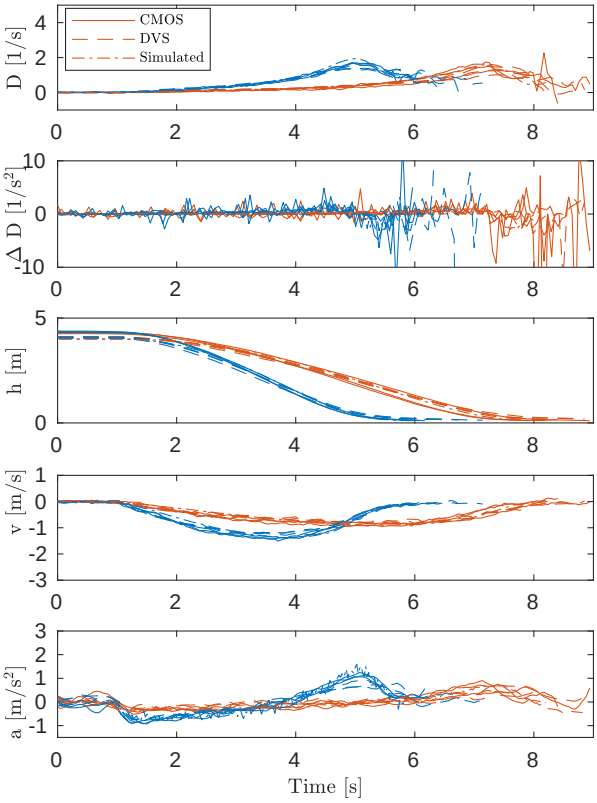


Figure 14: Vehicle states and observations from real world flights with the NN_1 (blue) and NN_2 (red) controllers. The results using the CMOS camera is shown in solid and the DVS in dashed. The simulated performance is also plotted in dot-dash for comparison.

Also notable is the repeatability of the landing maneuvers. The landings were performed three times each and each landing resulted in very similar trajectories. This shows that the evolutionary optimization converged to a robust solution as suggested by the analysis in Section 5.

9. Conclusion

This paper investigated the influence of abstraction of the sensory input to the reality gap for automatically optimized UAV agents tasked with performing quick yet safe landing. We have shown over multiple evolutionary runs and neu-

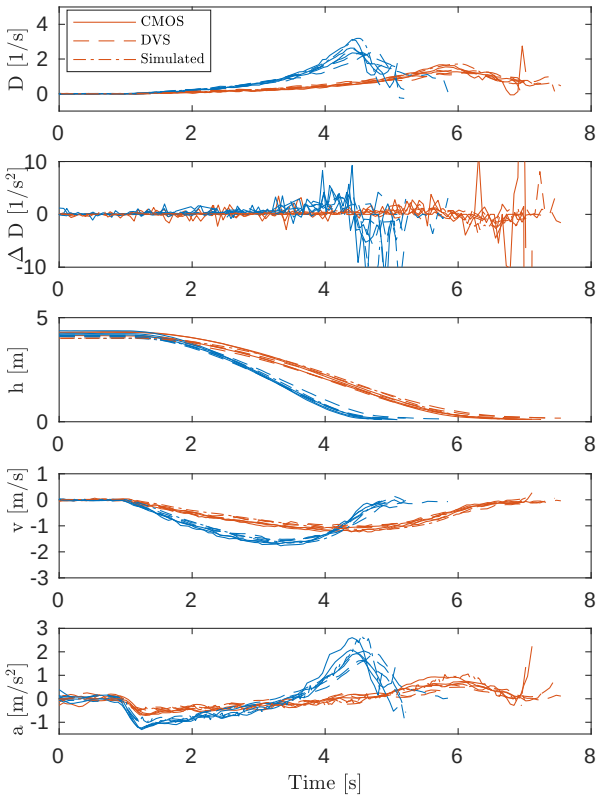


Figure 15: Vehicle states and observations from real world flights with the RNN_1 (blue) and RNN_2 (red). The results using the CMOS camera is shown in solid and the DVS in dashed. The simulated performance is also plotted in dot-dash for comparison.

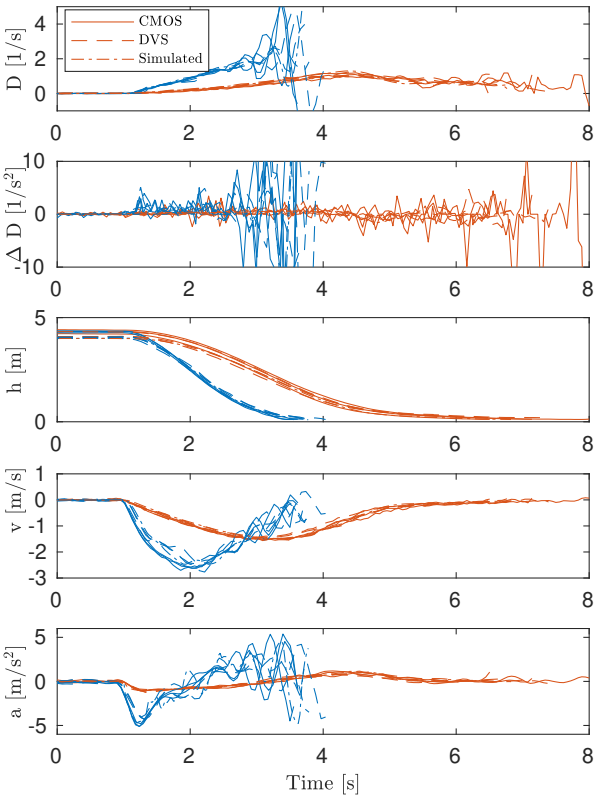


Figure 16: Vehicle states and observations from real world flights with the $CTRNN_1$ (blue) and $CTRNN_2$ (red). The results using the CMOS camera is shown in solid and the DVS in dashed. The simulated performance is also plotted in dot-dash for comparison.

rocontroller architectures, that abstraction does not unduly hamper the optimization power of the optimization as the agents developed a robust and effective method to land.

The optimized agents showed some landing strategies that were before not imagined by the human designers. One notable strategy is that instead of a simple proportional controller for the entire state space, an asymmetric response may be more appropriate to delay the onset of oscillations when performing the landing procedure. A strong response when the divergence error is positive and a weaker response when negative seems a good approach.

Tests in the real world showed the presence of significant differences between simulation and reality. The most significant was that the acceleration command tracking performance was poor, likely due to the drag and other non-linear aerodynamic effects which were not considered in simulation due to their modeling complexity. The resultant reality gap was crossed with the use of a closed loop controller representing another layer of abstraction, from the low-level raw motor control values to a desired vertical acceleration, which ensures robustness to the real world uncertainty.

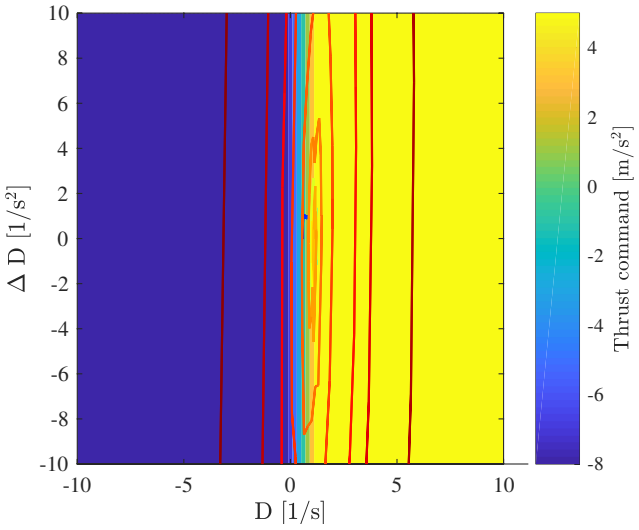
Finally, we showed that abstraction on the sensory input of the neurocontroller was robust to the reality gap when using two different input estimation techniques. Although two cameras with different imaging techniques were used, the resultant landing profile was very similar. Abstraction can therefore be a powerful tool when crossing the reality gap.

Future work could investigate how this approach can generalize to different vehicles operating in different real world environments. This would help to demonstrate the effectiveness of the approach to not only cross the reality gap but to address the more general transfer problem.

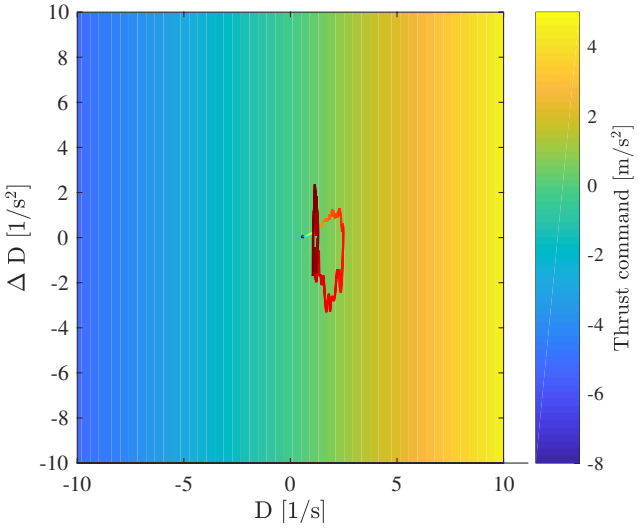
Acknowledgment

We would like to thank the team at Insightness AG for their assistance in porting the SEEM1 SDK to operate on the ARM processor of the Parrot Bebop 2. Without their assistance the results in the paper would not be possible.

Appendix: Controller Steady State Input-Output Mappings

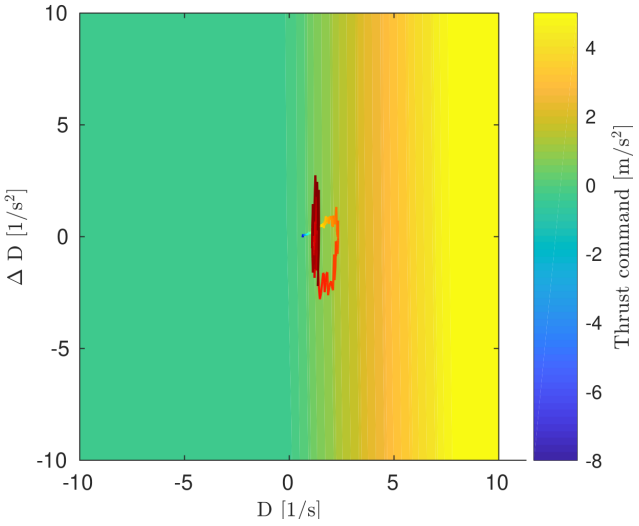


(a) Controller C_1

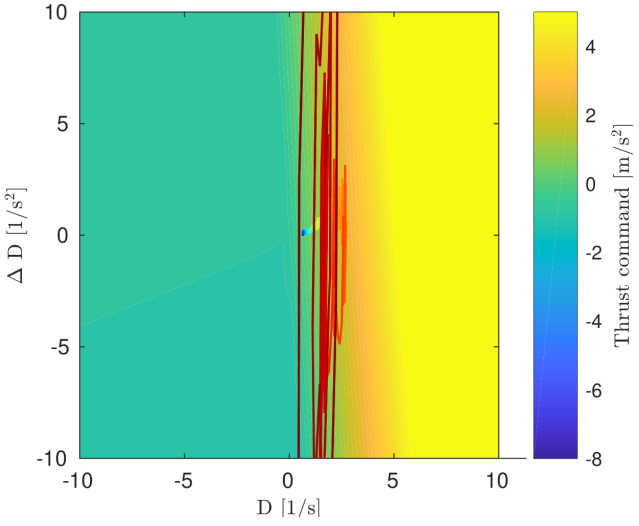


(b) Controller C_2

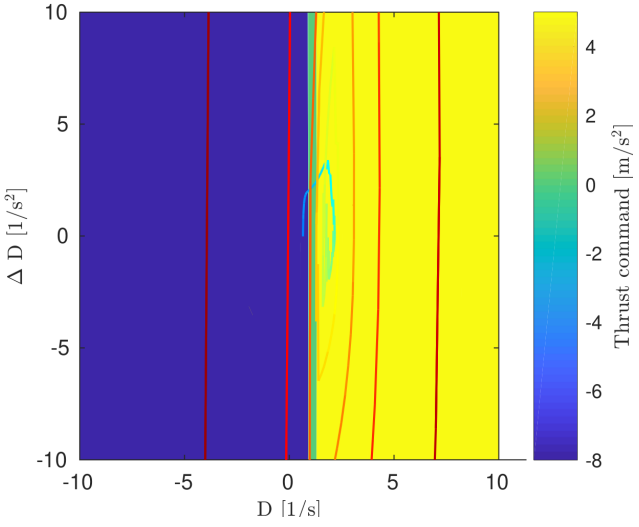
Figure 17: Steady state input-output mapping for hand designed controllers.



(a) NN_1

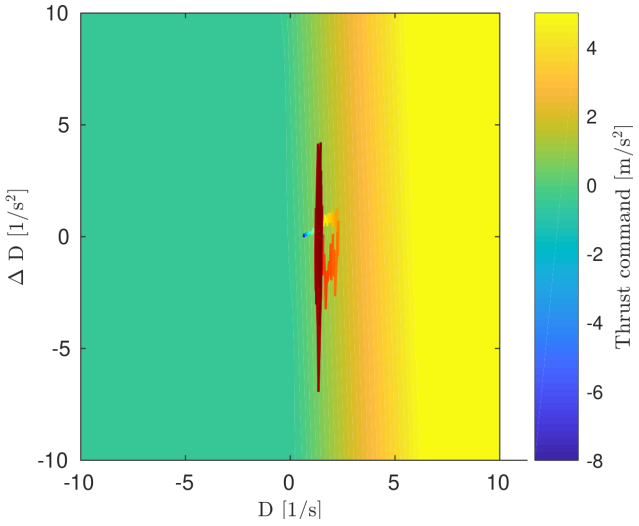


(b) NN_2

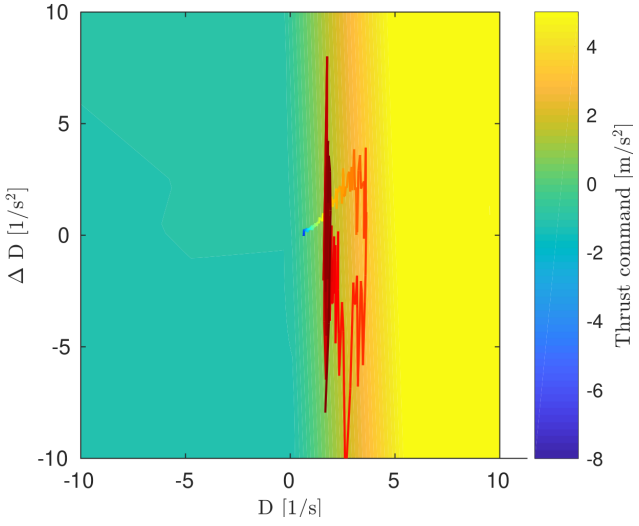


(c) NN_3

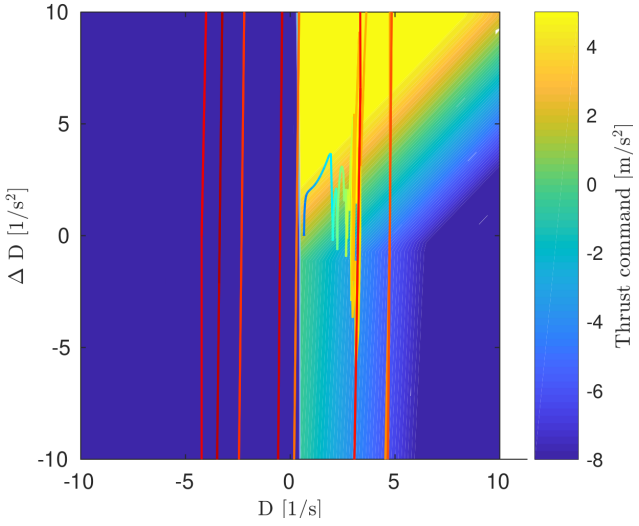
Figure 18: Steady state input-output mapping for NN controllers.



(a) RNN_1

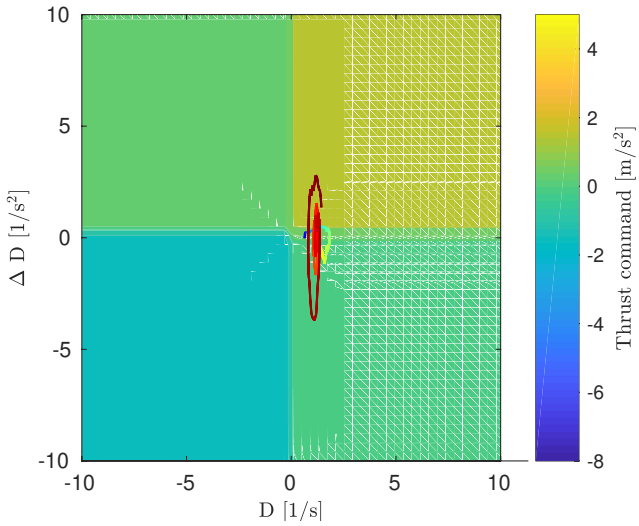


(b) RNN_2

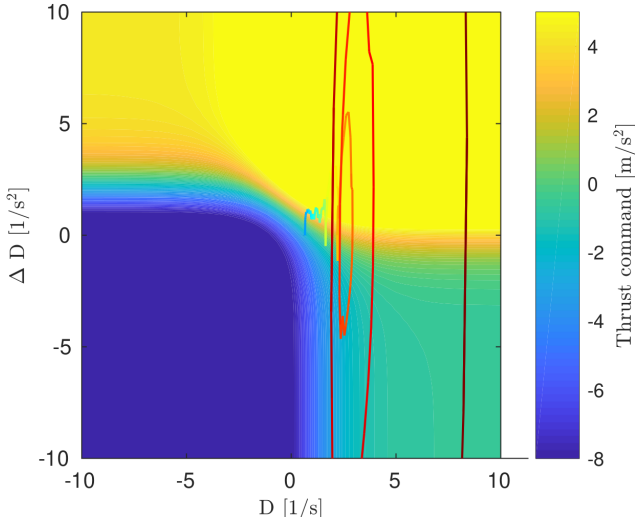


(c) RNN_3

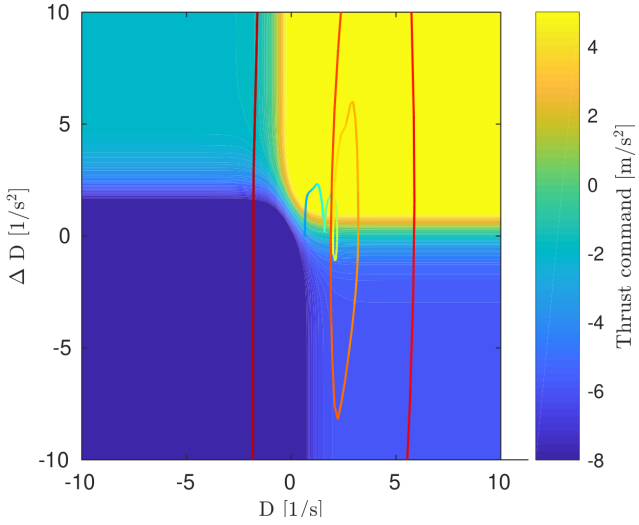
Figure 19: Steady state input-output mapping for NN controllers.



(a) $CTRNN_1$



(b) $CTRNN_2$



(c) $CTRNN_3$

Figure 20: Steady state input-output mapping for NN controllers.

References

[1] J. J. Gibson, The perception of the visual world, Boston, Houghton Mifflin, 1950

[2] E. Baird, N. Boeddekerb, M. R. Ibbotsonc, M. V. Srinivasan, A universal strategy for visually guided landing, Proceedings of the National Academy of Sciences 110 (46) (2013) 18686–18691 doi:10.1073/pnas.1314311110.

[3] B. Hérissé, T. Hamel, R. Mahony, F.-X. Russotto, Landing a VTOL Unmanned Aerial Vehicle on a Moving Platform Using Optical Flow, IEEE Transactions on Robotics 28 (1) (2012) 77–89 doi:10.1109/TRO.2011.2163435.

[4] G. C. H. E. de Croon, Monocular distance estimation with optical flow maneuvers and efference copies: a stability-based strategy, Bioinspiration & Biomimetics 11 (1) (2016) 1–30 doi:10.1088/1748-3190/11/1/016004.

[5] X. Clady, C. Clercq, S.-H. Ieng, F. Houseini, M. Randazzo, L. Natale, C. Bartolozzi, R. B. Benosman, Asynchronous visual event-based time-to-contact., Frontiers in neuroscience 8 (9) doi:10.3389/fnins.2014.00009.

[6] F. Kendoul, Four-dimensional guidance and control of movement using time-to-contact: Application to automated docking and landing of unmanned rotorcraft systems, International Journal of Robotics Research 33 (2) (2014) 237–267 doi:10.1177/0278364913509496.

[7] L. Rodolfo Garcia Carrillo, I. Fantoni, E. Rondón, A. Dzul, Three-dimensional Position and Velocity Regulation of a Quad-Rotorcraft Using Optical Flow, Transactions on Aerospace and Electronic Systems 51 (1) (2015) 358–371 doi:10.1109/TAES.2014.130607.

[8] A. Miller, B. Miller, A. Popov, K. Stepanyan, UAV Landing Based on the Optical Flow Videonavigation, Sensors 19 (6) (2019) 1351 doi:10.3390/s19061351.

[9] B. J. Pijnacker Hordijk, K. Y. W. Scheper, G. C. H. E. de Croon, Vertical landing for micro air vehicles using event-based optical flow, Journal of Field Robotics 35 (1) (2018) 69–90 doi:10.1002/rob.21764.

[10] H. W. Ho, G. C. H. E. de Croon, E.-J. van Kampen, Q. P. Chu, M. Mulder, Adaptive Gain Control Strategy for Constant Optical Flow Divergence Landing, IEEE Transactions on Robotics 34 (2) (2018) 508–516 doi:10.1109/TRO.2018.2817418.

[11] D. Howard, F. Kendoul, Towards Evolved Time to Contact Neurocontrollers for Quadcopters, in: Australasian Conference on Artificial Life and Computational Intelligence (ACALCI), 2016, pp. 336–347 doi:10.1007/978-3-319-28270-1.

[12] N. Jakobi, P. Husbands, I. Harvey, Noise and the reality gap: The use of simulation in evolutionary robotics, in: European Conference on Artificial Life, Springer, 1995, pp. 704–720

[13] S. Nolfi, D. Floreano, Evolutionary Robotics: The Biology, Intelligence and Technology, MIT Press, Cambridge, MA, 2000

[14] J. C. Bongard, Evolutionary Robotics, Communications of the ACM 56 (8) (2013) 74–83 doi:10.1145/2492007.2493883.

[15] N. Jakobi, P. Husbands, I. Harvey, Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics, in: Advances in Artificial Life, Springer Berlin Heidelberg, Granada, 1995, pp. 704–720 doi:10.1007/3-540-59496-5_337.

[16] S. Koos, J.-B. Mouret, S. Doncieux, Crossing the reality gap in evolutionary robotics by promoting transferable controllers, in: Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10, 2010, p. 119 doi:10.1145/1830483.1830505.

[17] P. Szczawinski, M. Duarte, S. M. Oliveira, A. L. Christensen, Toward Evolved Vision-based Control for a Quadrocopter, Proceedings of the 9th Conference on Telecommunications (CONFTELE) (2013) 153–156

[18] K. Y. W. Scheper, G. C. H. E. de Croon, Abstraction as a Mechanism to Cross the Reality Gap in Evolutionary Robotics, in: From Animals to Animats 14: Proceedings of the 14th International Conference on Simulation of Adaptive Behavior (SAB2016), Vol. 9825 LNCS, Springer International Publishing, Aberystwyth University, Wales, UK, 2016, pp. 280–292 doi:10.1007/978-3-319-43488-9_25.

[19] K. Y. W. Scheper, G. C. H. E. de Croon, Abstraction, Sensory-Motor Coordination, and the Reality Gap in Evolutionary Robotics, Artificial Life 23 (2) doi:10.1162/ARTL_a_00227.

[20] G. C. H. E. de Croon, H. W. Ho, C. De Wagter, E.-J. van Kampen, B. D. W. Remes, Q. P. Chu, Optic-flow based slope estimation for autonomous landing, International Journal of Micro Air Vehicles 5 (4) (2013) 287–297 doi:10.1260/1756-8293.5.4.287.

[21] H. W. Ho, G. C. H. E. de Croon, Characterization of Flow Field Divergence for MAVs Vertical Control Landing, in: AIAA Guidance, Navigation, and Control Conference, 2016, pp. 1–13 doi:10.2514/6.2016-0106.

[22] F. Paredes Vallés, Neuromorphic Computing of Event-Based Data for High-Speed Vision-Based Navigation, Msc, Delft University of Technology

[23] H. C. Longuet-Higgins, K. Prazdny, The interpretation of a moving retinal image, Proceedings of the Royal Society of London Biological Sciences 208 (1173) (1980) 385–397 doi:10.1098/rspb.1980.0057.

[24] G. Hattenberger, M. Bronz, M. Gorraz, Using the Paparazzi UAV System for Scientific Research, in: International Micro Air Vehicle Conference and Competition 2014, IMAV, Delft, Netherlands, 2014, pp. 247–252 doi:10.4233/uuid:b38fbdb7-e6bd-440d-93be-f7dd1457be60.

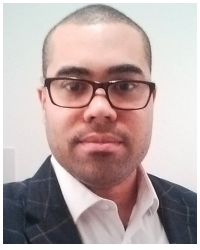
[25] B. D. Lucas, T. Kanade, An Iterative Image Registration Technique with an Application to Stereo Vision, in: Proceedings of Imaging Understanding Workshop, 1981, pp. 121–130

[26] F. van Breugel, K. Morgansen, M. H. Dickinson, Monocular distance estimation from optic flow during active landing maneuvers, Bioinspiration & Biomimetics 9 doi:10.1088/1748-3182/9/2/025002.

[27] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197 doi:10.1109/4235.996017.

[28] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, C. Gagné, DEAP: Evolutionary Algorithms Made Easy, Journal of Machine Learning Research 13 (2012) 2171–2175 doi:http://www.jmlr.org/papers/v13/fortin12a.html.

[29] R. D. Beer, On the Dynamics of Small Continuous-Time Recurrent Neural Networks, Adaptive Behavior 3 (4) doi:10.1177/105971239500300405.



Kirk Y. W. Scheper received his M.Sc. and Ph.D from the Micro Air Vehicle Laboratory in the Faculty of Aerospace Engineering at Delft University of Technology, the Netherlands, in 2014 and 2019 respectively. His research focuses on the development of embedded software which facilitates high level autonomy of micro air vehicles. His work is mainly in the fields of evolutionary robotics, embodied cognition, and vision-based navigation and event-based cameras.

Guido C. H. E. de Croon received his M.Sc. and Ph.D. in the field of Artificial Intelligence at Maastricht University, the Netherlands. His research interest lies with computationally efficient algorithms for robot autonomy, with an emphasis on computer vision. Since 2008 he has worked on algorithms for achieving autonomous flight with small and light-weight flying robots, such as the DelFly flapping wing MAV. In 2011-2012, he was a research fellow in the Advanced Concepts Team of the European Space Agency, where he studied topics such as optical flow based control algorithms for extraterrestrial landing scenarios. Currently, he is associate professor at Delft University of Technology, the Netherlands, where he is the scientific lead of the Micro Air Vehicle Laboratory.