Dragonfly Missile Command: Neuromorphic Controllers for Interception and Flight

M. Elijah Wangeman
BS Computer Science†
maw4678@rit.edu

Diana Velychko
MS Artificial Intelligence†
dv6943@rit.edu

Jake Streamer
BS Individualized Study†
jws8495@rit.edu

Nithyasri Narasimhan
MS Artificial Intelligence†
nn7985@rit.edu

†*Rochester Institute of Technology*
Rochester, NY, USA

***Abstract*—Dragonflies are uniquely effective, intelligent, and efficient at navigating a 3D environment to intercept a target. They reliably fly on an interception course unique across all observed pursuit predators. The dragonfly achieves this task with a brain of only 1 million neurons, seemingly computing trajectory prediction on as few as tens of thousands, through just 3 neural layers. The AKD1000 is a neuromorphic processor that can implement Spiking Neural Networks (SNN) on hardware. By modeling a dragonfly pursuit and training a Convolutional Neural Network (CNN) model to control the dragonfly purely from visual stimulus based on research from Dr. Chance [1], then converting that model to a Akida compatible SNN, we evidence the potential of edge neuromorphic devices for spatial navigation and targeting.**

I. Introduction

Recent literature has become perhaps unusually interested in the mind of the common dragonfly. The dragonfly is a carnivorous insect that hunts and eats smaller insects. Zoological studies indicate that dragonflies catch approximately 95% of the prey they seek in the air - a remarkable success rate. [2] This success rate is enabled by their speed and maneuverability, but also their impressive (and unique among insects) ability to chase prey on an intercepting trajectory, accurately predicting prey trajectory. Notably, they react to alterations in their prey's trajectory within 30-50 ms, demonstrating a maximum of 3 layers of neurons of processing are required for the dragonfly to make motor decisions, in the process predicting the path of their prey.

Anti-ballistic missile systems have long relied on complex, computationally expensive, mathematics to intercept flying projectiles. These missile systems, while advanced, require enormous resources to fire and then to apply advanced position correction algorithms on scarce integrated systems hardware.

Drone interception is a more appropriate analogy to dragonfly interception. Unmanned Aerial Vehicles (UAVs) are exploding in research and development for a wide variety of applications. There is excellent potential for new, fast, power-efficient UAVs utilizing bioinspired, neuromorphic hardware systems.

Applications extend beyond these more exciting examples; modern camera drones, like those by DJI, rely heavily on machine vision for tracking subjects, a computationally expensive effort that is consequential on battery powered, range-limited consumer drones.

II. Methodology

Drones have limitations in weight, storage, and energy as these resources are sparse. Neuromorphic chip architecture permits to reduce the energy consumption and allows to store some information with the use of time dimension. In our experiment, we use the AKD1000 chip, which is a neuromorphic processing chip developed by BrainChip Holdings. Unlike traditional GPUs, which rely on dense matrix operations, the AKD1000 processes sparse, event-driven data, making it much more efficient for real-world, event-based workloads. This approach not only reduces computational overhead but also aligns closely with the operation of the human brain, offering a pathway to more biologically inspired AI. The main reason to use the neuromorphic chip is its energy efficiency, which is crucial in edge devices such as UAVs.

Dragonfly brains have some unique characteristics that make models of them more straightforward than might initially be expected. Distinct neural pathways exist directly between the eyes and wings of a dragonfly, seemingly bypassing it's central nervous system entirely. This is theorized to be responsible for their fast reaction times while chasing, and points towards a relatively simple but very fast neural network we have to simulate: only considering that one pathway, ignoring the rest of the dragonfly nervous system, but still having an intelligent navigation model.

Dragonflies have some advantages, particularly in visual processing speed (200 hz) and a near 160° FOV[1] Research indicates, though, this comes at the cost of visual detail and depth perception. The lack of depth perception makes their trajectory predicting powers even more impressive, but also points to it not being a computationally complex 3D space trajectory solver, but instead some more novel (and, importantly, efficient) evolved approach of guaranteeing a collision course.

A. Neural network

```
X = np.load('data/X.npy')
y = np.load('data/y.npy')
```

[1]Which, in the interest of reasonable precision in tracking targets within a FOV, we are not replicating in our models.

```
model = Sequential([
    Input(shape=(FOV_SIZE, FOV_SIZE, 1)),
    Flatten(),
    Dense(441, activation='relu'),
    Dense(194481, activation='relu'),
    Dense(441, activation='relu'),
    Dense(2, activation='linear'),  # Output layer
    ↪  for pitch and yaw adjustments
    Reshape((2,))
])

model.compile(optimizer=Adam(learning_rate=0.001),
↪  loss='mse', metrics=['accuracy'])

checkpoint =
↪  ModelCheckpoint('models/algo_trained.keras',
↪  monitor='val_loss', save_best_only=True,
↪  mode='min')

model.fit(X, y, epochs=10, batch_size=32,
↪  validation_split=0.2, callbacks=[checkpoint])

model.save('models/algo_trained_final.keras')
```

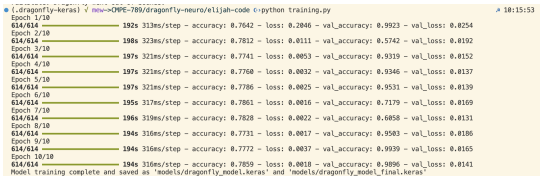

Fig. 1. `algo_model` training that would've taken upwards of 3 hours per epoch on google colab.

The proposed neural network mimics a dragonfly's prey interception system with a unique approach to tracking and capturing prey. It consists of four dense layers presented in the code snippet II-A.The first layer consists of 441 neurons representing the dragonfly's visual field, arranged in a 21 by 21 square array. [1] This layer captures the prey's location within the dragonfly's field of view. In the second layer, we used 21^4 (194481) neurons for the creation of the movement directions. A third set of 441 neurons hypothetically helps determine which neurons should be aligned with the prey's image. The final layer consists of 2 neurons that output yaw and rotation.

The network is first setup as an artificial neural network and then converted into the spiking neural network using Keras conversion function.

B. Keras model → Akida SNN

```
model_keras =
↪  keras.models.load_model('models/algo_trained.keras')

qparams = QuantizationParams(input_weight_bits=8,
↪  weight_bits=8, activation_bits=8)
model_quantized = quantize(model_keras,
↪  qparams=qparams)
model_quantized.summary()

model_akida = convert(model_quantized)
model_akida.summary()

model_akida.save('models/algo_trained_akida.fbz')
```

III. Experimental Design

The algorithms and a neural network are tested in the 3D space within a scenario presented in the code snippet III-A.

A. Simulated environment

```
class Scenario:
    def __init__(self, prey_trajectory,
    ↪  initial_dragonfly_pos,
    ↪  initial_dragonfly_heading,
    ↪  brain=brain_classic_direct):
        self.time = 0
        self.prey_trajectory = prey_trajectory
        self.brain = brain

        # Dragonfly initial state
        self.dragonfly_pos = initial_dragonfly_pos
        self.dragonfly_heading =
        ↪  initial_dragonfly_heading

        # Initialize dragonfly trajectory with
        ↪  starting position
        self.dragonfly_trajectory =
        ↪  np.array([initial_dragonfly_pos])
```

We generate linear and parabolical prey trajectories for testing the algorithm in 3D space. From a predefined initial point, we generate each step of the dragonfly using our algorithm. In the first state, we rotate the dragonfly such that its field of view captures the prey position. The main idea is to place the prey approximately in the center of its visual field. The field of view is set up to be a 2D space with a width and height of 21 pixels. [1]

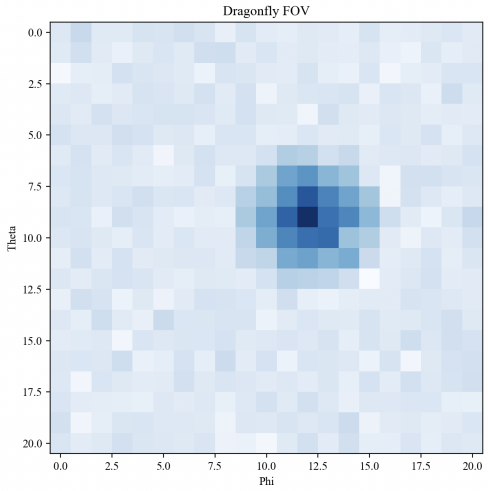

Fig. 2. The dragonfly perspective computed from dragonfly and prey location and dragonfly heading by the code in VIII-A.

The code snippet VIII-A presents the calculation of the field of view. For the sake of this experiment, we test two brain mechanics to generate a field of view: classical calculation and calculation with offset based on the previous step. The code for a classical brain is presented in code snippet VIII-B, and the code for a brain for calculation with the offset based on the previous step in VIII-C.

B. Dragonfly decision-making

For the experimental setup, we simulate the scenario in which the prey trajectory is predefined and unfolds step-by-step. For each step of the dragonfly, we predict an ideal yaw and rotation through whatever brain function was selected when the scenario was initialized. In each time step, we evaluate the position of both actors, and if the dragonfly is closer to prey than a predefined threshold (< 0.2), the prey is caught, and it is considered a winning scenario. If the dragonfly or prey flies outside the bounds of the 3D space generated for the simulation, or the timestep reaches the end of the array of prey positions (50 for our tests), it is regarded as a failed attempt. Figure 1 shows a winning attempt to catch the prey in a simulated environment.

We collect all winning trajectories and train the neural network with them. As a result, the network is capable of learning the dependencies between the position of the prey in comparison to the position of the hunter and determining the winning combination of yaw and rotation. We use SNN to generate the rotation and yaw for each step of the dragonfly.

IV. Analysis

Table I provides an overview of a subset of our approaches and their outcomes. One somewhat unexpected result was the capability of the simple discrete proportional control algorithm in capturing the prey; for very low computational cost, it competes within 0.2 of the success rate of both our and Dr. Chance's dragonflies, and within 0.25 of the success rate of actual dragonflies.[2] Training on pruned datasets enabled some measure of improvement, but stayed within 0.08 of the classical results. It's worth mentioning that the increase from the floating point algo_trained keras model on M1 Max to it's quantized sibling on the AKD1000 is just chance- the results skew 3-4% across tests.[3]

The results of the on-chip neuromorphic inferencing are ideal and further evidence the value of this project. 13.42 ms is a decision time that makes real-time navigation on device not just feasible, but currently implementable. It does this while losing no performance capabilities quantized to 8 bits and comfortably fitting on the AKD1000. These are very compelling results considering how little we have optimized the models for that silicon, coming from a training infrastructure based on Dr. Chances research and for the most part developed without special considerations for the AKD1000.

Comparing 'algo_trained_final' and 'algo_trained' provides a very clear example of overfitting. The 'rand_final' model puts into perspective how intelligent and successful even relatively uncompetitive algorithms like our offset fov approach are, which was an attempt at replicating Dr. Chances second layer of neurons that enabled offset targeting, which allows predictive instead of pursuit interceptions. Unfortunately, our non-neural approach significantly outperforms the direct pursuit algorithm 25% of the time[4] and significantly underperforms 75% of the time.

A. AKD1000 Efficiency

```
Average framerate = 76.25 fps
Last inference power range (mW):  Avg 1027.20 / Min
↪  915.00 / Max 1142.30 / Std 192.65
Last inference energy consumed (mJ/frame): 13.25
```

The efficiency we can see in the AKD1000 running our quantized and converted model is impressive, especially considering it is outperforming a far larger, more expensive, and less efficient M1 Max chip 13.25 to 59.42 ms.

V. Limitations

The assumption of time independence also oversimplifies biological systems that rely on temporal dynamics, such as spike timing in SNNs, such as the ones found in human neurons. This simplification significantly undermines the potential of the model's implementation in real-world situations as well as its competition with biology. It also limits the system to basic behaviors while neglecting advanced dynamics such as adaptive learning. Introducing complex multi-agent scenarios would not improve anything, considering that the framework would not be ideal. Without completely utilizing the hardware capabilities of the Akida, the translation of this concept to SNNs remains unclear.

The current approach of a constant speed and discrete turn steps, mimicking Dr. Chance's research, must be optimized either for precision or speed in turning. Our current turning speed[5] of $\frac{\pi}{12}$ allows the dragonfly adequate precision to aim for intercept accurately, but also prevents it from reacting quickly when the prey is near, and any orthogonal approaches risk the prey moving out of the dragonflies FOV faster than the dragonfly can turn to maintain that tracking, as seen in 4. Fixing this is potentially negligible- already, the outputs of the various model are being clipped and quantized in order to match the requirements of `change_heading()`. Obviously this doesn't work now[6], but the sole limit from repeating the same setup with floating point outputs is a non-NN brain that has useful floating point outputs.

On silicon inferencing greatly outperforms even the M1 Max chip runnng the non-quantized models, and with no loss in accuracy. In fact, the 13 millisecond average decision times actually outpace the 30-50 ms reaction times of dragonflies. This is still dwarfed by the speed of the classical approach, which computes in fractions of a millisecond. Although the several magnitudes of processing speed lost may seem a poor exchange for a 0.08 increase, we believe that we are far from the limits of the neural network, especially as we move beyond relying on the classical control algorithms for training data.

[2]Don't take this comparison seriously, neither model is equivalent to actual dragonfly hunting conditions.

[3]These tests were ran with the `count_win_fail_states()` function, visible within the Scenario class on the Github.

[4]This isn't a coincidence, just a result of the number of cardinal directions.

[5]In reality, the rotational translation per timestep.

[6]It was trained on a discretely turning slgorithm, after all.

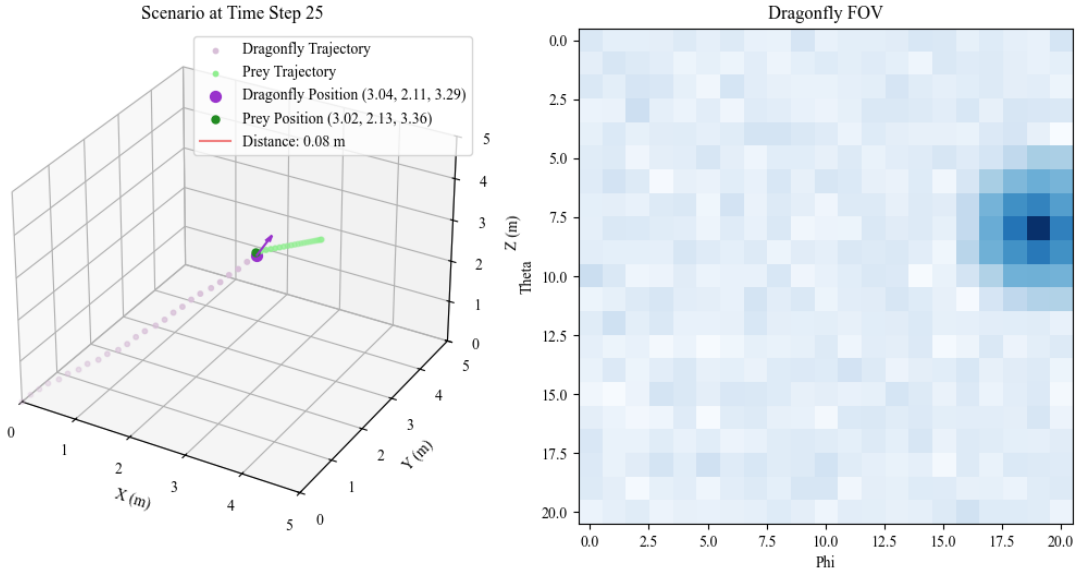

Fig. 3. Winning outcome in the simulated environment.

TABLE I
COMPARISON OF APPROACHES BASED ON BRAIN, DECISION TIME, AND SUCCESS RATE.

Approach	Brain	Model	Decision Time (ms)	Success Rate
Akida	`brain_akida()`	`algo_trained_quantized`	13.42	0.82
Keras	`brain_keras()`	`algo_trained`	59.42	0.81
Algorithm	`brain_classic_direct()`	N/A	0.003344	0.72
Keras	`brain_keras()`	`algo_trained_final`	70.52	0.61
Algorithm	`brain_offset_prev()`	N/A	0.0066	0.46
Keras	`brain_keras()`	`rand_final`	63.21	0.01

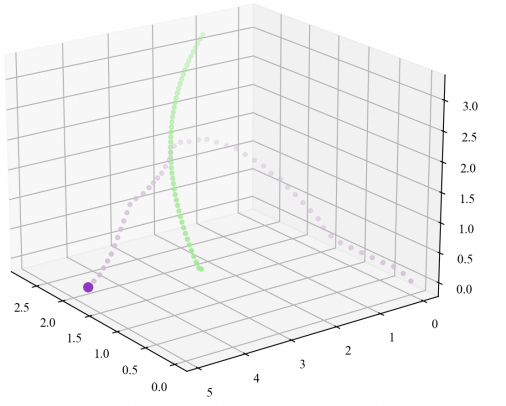

Fig. 4. Dragonfly on orthogonal approach misses prey, meandering off guided by noise.

VI. Further Applications

We want to solve for ideal interception trajectories between a given dragonfly starting state and a prey trajectory mathematically (without considering the FOV) based on a set limitation of the dragonfly's movement[7] This is trivially implemented with a backtracking algorithm, from which paired FOV's & motor responses could be produced to train a model independently of our control code.

The modular nature of our codebase provides us a number of directions in which to expand this research beyond gradual improvement of the current dragonfly model. Time-dependency, obstacle avoidance, floating point motor control, variable speed, independent training, and unpredictable prey trajectory are all alterations to our existing experiments we have considered and are interested in pursuing.

VII. Conclusion

The success of our dragonfly pursuit models corroborates Dr. Chance's research and extends her insights into on-chip applications, providing valuable evidence towards the feasibility of lightweight and efficient visual neuromorphic control systems for a variety of devices that navigate spatial environments. We established a strong foundation in the existing model and it's hardware implementation which leverages future research towards high efficiency spatial pursuit SNN. The success of our models are responsive to gradual improvement, and there are a number of unturned stones that could dramatically improve the performance and breadth of applications of our research.

A. Github

GitHub Link

[7]Like, the 0.15 m per timestep and $\frac{\pi}{12}$ per timestep rules.

VIII. Appendix

A. Dragonfly FOV code

```
def calculate_fov(dragonfly_heading, dragonfly_pos,
↪  prey_pos, fov_size=21, fov_angle=np.pi):
    fov = np.zeros((fov_size, fov_size))

    relative_pos = prey_pos - dragonfly_pos

    # spherical coordinates
    r = np.linalg.norm(relative_pos)
    theta = np.arccos(relative_pos[2] / r) if r != 0
    ↪  else 0
    phi = np.arctan2(relative_pos[1],
    ↪  relative_pos[0])

    heading_theta, heading_phi = dragonfly_heading

    # diff between heading and prey xyz
    delta_theta = theta - heading_theta
    delta_phi = phi - heading_phi

    # normalize
    if np.abs(delta_theta) <= fov_angle / 2 and
    ↪  np.abs(delta_phi) <= fov_angle / 2:
        # map to FOV array
        i_center = int((delta_theta + fov_angle / 2)
        ↪  / fov_angle * (fov_size - 1))
        j_center = int((delta_phi + fov_angle / 2) /
        ↪  fov_angle * (fov_size - 1))

        # intensity from distance (test removed)
        intensity = max(0.5, min(1, 1 - (r / 5)))

        # blur
        spread = max(1, int((1 - r / 5) * (fov_size
        ↪  // 4)))
        for di in range(-spread, spread + 1):
            for dj in range(-spread, spread + 1):
                i = i_center + di
                j = j_center + dj
                if 0 <= i < fov_size and 0 <= j <
                ↪  fov_size:
                    distance_factor = max(0, 1 -
                    ↪  (np.sqrt(di**2 + dj**2) /
                    ↪  spread))
                    fov[i, j] += intensity *
                    ↪  distance_factor

    # add noise (only up to 0.05) to fov:
    fov += np.random.normal(0, NOISE, fov.shape)

    return fov
```

B. Brain functions

```
def brain_keras(fov):
    fov = np.expand_dims(fov, axis=0)
    fov = np.expand_dims(fov, axis=-1)

    prediction = MODEL.predict(fov, verbose=0)
    pitch, yaw = prediction[0]

    # quantize to -1, 0, or 1
    pitch = int(np.round(pitch))
    yaw = int(np.round(yaw))

    # clip
    pitch = np.clip(pitch, -1, 1)
    yaw = np.clip(yaw, -1, 1)

    return (pitch, yaw)

akida_model = Model("path_to_your_akida_model.fbz")
core = Core()
core.load_model(akida_model)

def brain_akida(fov):
    ...
    prediction = core.infer(fov)
    ...
    return (pitch, yaw)

def brain_classic_direct(fov):
    fov_size = fov.shape[0]
    center = fov_size // 2

    max_index = np.unravel_index(np.argmax(fov),
    ↪  fov.shape)
    max_i, max_j = max_index

    if max_i < center:
        pitch = -1
    elif max_i > center:
        pitch = 1
    else:
        pitch = 0

    if max_j < center:
        yaw = -1
    elif max_j > center:
        yaw = 1
    else:
        yaw = 0

    return (pitch, yaw)
```

C. Offset Brain Function

```
def brain_offset_prev(fov, prev_fov=None):
    fov_size = fov.shape[0]
    center = fov_size // 2

    # Find the indices of the maximum value in the
    ↪  current FOV
    max_index = np.unravel_index(np.argmax(fov),
    ↪  fov.shape)
    max_i, max_j = max_index

    # If previous FOV is provided, calculate
    ↪  movement prediction
    if prev_fov is not None:
        prev_max_index =
        ↪  np.unravel_index(np.argmax(prev_fov),
        ↪  prev_fov.shape)
        prev_i, prev_j = prev_max_index

        # Predict movement direction
        i_delta = max_i - prev_i
        j_delta = max_j - prev_j

        # Predict next position with some
        ↪  anticipation
        predicted_i = max_i + i_delta
        predicted_j = max_j + j_delta

        # Determine pitch adjustment based on
        ↪  predicted position
        if predicted_i < center - fov_size * 0.1:
            pitch = -1
        elif predicted_i > center + fov_size * 0.1:
            pitch = 1
        else:
            pitch = 0

        # Determine yaw adjustment based on
        ↪  predicted position
        if predicted_j < center - fov_size * 0.1:
            yaw = -1
        elif predicted_j > center + fov_size * 0.1:
```

```
            yaw = 1
        else:
            yaw = 0

    else:
        # If no previous FOV, use current FOV
        ↪ position
        if max_i < center - fov_size * 0.1:
            pitch = -1
        elif max_i > center + fov_size * 0.1:
            pitch = 1
        else:
            pitch = 0

        if max_j < center - fov_size * 0.1:
            yaw = -1
        elif max_j > center + fov_size * 0.1:
            yaw = 1
        else:
            yaw = 0

    return (pitch, yaw)
```

REFERENCES

[1] S. N. Fry, R. Sayaman, and M. A. Frye, "Interception from a dragonfly neural network model," in *Proceedings of the 2020 International Conference on Neuromorphic Systems*, 2020. [Online]. Available: https://dl.acm.org/doi/10.1145/3407197.3407218

[2] F. Chance, "Fast, efficient neural networks copy dragonfly brains," *IEEE Spectrum*, 2021. [Online]. Available: https://spectrum.ieee.org/fast-efficient-neural-networks-copy-dragonfly-brains

[3] F. S. Chance, "Interception from a dragonfly neural network model," in *International Conference on Neuromorphic Systems 2020*, ser. ICONS 2020. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3407197.3407218

[4] H.-T. Lin, I. Siwanowicz, and A. Leonardo, "Wireless recordings from dragonfly target detecting neurons during prey interception flight," *bioRxiv*, 2024. [Online]. Available: https://www.biorxiv.org/content/early/2024/11/13/2024.11.12.622977

[5] H. Ma, P. Gong, Y. Tian, Q. Wu, M. Pan, H. Yin, Y. Liu, and C. Chen, "Hifly-dragon: A dragonfly inspired flapping flying robot with modified, resonant, direct-driven flapping mechanisms," *Drones*, vol. 8, no. 4, 2024. [Online]. Available: https://www.mdpi.com/2504-446X/8/4/126

[6] P. Arena, M. Calí, L. Patané, A. Portera, and A. G. Spinosa, "A cnn-based neuromorphic model for classification and decision control," *Nonlinear Dynamics*, vol. 95, p. 1999–2017, 2019. [Online]. Available: https://link.springer.com/article/10.1007/s11071-018-4673-4

[7] S. A. Combes, "Dragonflies predict and plan their hunts," *Nature*, vol. 517, no. 7534, pp. 279–280, 2015.

[8] P. T. Gonzalez-Bellido, H. Peng, J. Yang, A. P. Georgopoulos, and R. M. Olberg, "Eight pairs of descending visual neurons in the dragonfly give wing motor centers accurate population vector of prey direction," *Proceedings of the National Academy of Sciences*, vol. 110, no. 2, pp. 696–701, 2013.

[9] M. Mischiati, H.-T. Lin, P. Herold, E. Imler, R. Olberg, and A. Leonardo, "Internal models direct dragonfly interception steering," *Nature*, vol. 517, no. 7534, pp. 333–338, 2015.