Follow Anything: Open-set detection, tracking, and following in real-time

Alaa Maalouf[1,3,*], *Member, IEEE,* Ninad Jadhav[2,3], Krishna Murthy Jatavallabhula[1], Makram Chahine[1], *Member, IEEE,* Daniel M.Vogt[2,3], Robert J. Wood[2,3], *Fellow, IEEE,* Antonio Torralba[1,3], and Daniela Rus[1,3], *Fellow, IEEE*

***Abstract*—Tracking and following objects of interest is critical to several robotics use cases, ranging from industrial automation to logistics and warehousing, to healthcare and security. In this paper, we present a robotic system to detect, track, and follow any object in real-time. Our approach, dubbed *follow anything* (*FAn*), is an open-vocabulary and multimodal model – it is not restricted to concepts seen at training time and can be applied to novel classes at inference time using text, images, or click queries. Leveraging rich visual descriptors from large-scale pre-trained models (*foundation models*), *FAn* can detect and segment objects by matching multimodal queries (text, images, clicks) against an input image sequence. These detected and segmented objects are tracked across image frames, all while accounting for occlusion and object re-emergence. We demonstrate *FAn* on a real-world robotic system (a micro aerial vehicle), and report its ability to seamlessly follow the objects of interest in a real-time control loop. *FAn* can be deployed on a laptop with a lightweight (6-8 GB) graphics card, achieving a throughput of 6-20 frames per second. To enable rapid adoption, deployment, and extensibility, we open-source our code on our project webpage. We also encourage the reader to watch our 5-minute explainer video.**

***Index Terms*—AI-Enabled Robotics; Semantic Scene Understanding; Object Detection, Segmentation and Categorization.**

I. Introduction

DETECTING, tracking, and following objects of interest is critical to several robotics use-cases, such as industrial automation, logistics and warehousing, healthcare, and security [1]–[4]. Notably, one of the key drivers of continuous progress in providing robust object-following systems is the combination of computer vision and deep learning [5], [6], where training deep convolutional networks on large labeled datasets have made tremendous strides in this area. Specifically, the object following task relies on the video segmentation and tracking task, which can be categorized into distinct subtasks. These include interactive (scribble or click-based) video segmentation [7], where a user draws a box around or clicks on the object to segment and track, mask-guided video segmentation [8]–[11], which assumes the presence of a mask to track, and automatic video segmentation [12]–[16], which assumes that the user does not interact with the algorithm to obtain the segmentation masks; methods should provide a set of object candidates with no overlapping pixels that span through the video sequence, however, these candidates are not specific, meaning that the segmentation will be applied to all of the seen objects, and not recognize the desired object. Thus, to automatically identify the required object to follow, numerous detection approaches have been suggested [17], [18] such as RCNN and its variants [19]–[21], YOLO and its variants [22]–[24], and more [25], [26]. However, existing robotic systems for object detection and following suffer two notable shortcomings: (i) They are *closed-set*, i.e., the set of objects to detect and follow is assumed to be available a priori (during the training phase). Thus, such systems are only able to handle a *fixed set of object categories* [2], [27]–[30], limiting their adaptability; adapting to newer object categories necessitates finetuning the model. (ii) Additionally, the objects of interest are specified (queried) only by a class label, which is *often unintuitive for end-users* to specify, imposing restrictions on how users interact with the system [2], [31], [32].

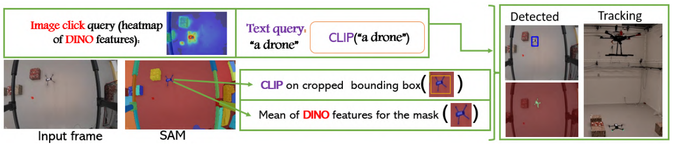


Image click query (heatmap of DINO features):
Text query: "a drone"
CLIP("a drone")
CLIP on cropped bounding box( )
Mean of DINO features for the mask ( )
Input frame
SAM
Detected
Tracking

Fig. 1: **Follow anything** (*FAn*) is a real-time robotic system to detect, track, and follow objects in an open-vocabulary setting. Objects of interest may be specified using text, images, or clicks. *FAn* leverages foundation models like CLIP [33], DINO [34], and SAM [35] to compute segmentation masks that best align with the queried objects. These objects are tracked across video frames while accounting for occlusion and object re-emergence; enabling real-time following of objects of interest by a robot platform.

Deep learning is currently undergoing another wave of evermore performant and robust model design, with the creation of increasingly big and multimodal models trained on internet scale amount data containing billions of images, text, and audio. These highly capable models (e.g., CLIP [33], DINO [34]) have demonstrated impressive performance in open-set scenarios (i.e., the objects of interest are only supplied at inference time, and not trained for a specific task) [36], [37]. Notably, recent robotics approaches using foundation models

Manuscript received: July 21, 2023; Revised December 1, 2023; Accepted January 29, 2024. This paper was recommended for publication by Editor Markus Vincze upon evaluation of the Associate Editor and Reviewers' comments. This study was funded by Project CETI via grants from Dalio Philanthropies and Ocean X; Sea Grape Foundation; Virgin Unite, Rosamund Zander/Hansjorg Wyss, Chris Anderson/Jacqueline Novogratz through The Audacious Project: a collaborative funding initiative housed at TED. This research was supported in part by the AI2050 program at Schmidt Futures (grant G-22-63172). Digital Object Identifier (DOI): see top of this page.

[1] Alaa Maalouf, Krishna Murthy Jatavallabhula, Makram Chahine, Antonio Torralba, and Daniela Rus are with the Computer Science and Artificial Intelligence Lab, MIT, Cambridge, MA 02139 USA. [2] Ninad Jadhav, Daniel M.Vogt, and Robert J. Wood are with the John A. Paulson School Of Engineering And Applied Sciences, Harvard, Boston, MA 02134 USA. [3] Alaa Maalouf, Ninad Jadhav, Daniel M.Vogt, Robert J. Wood, Antonio Torralba, and Daniela Rus are with Project CETI, New York, NY, 10003 USA. [*] Correspondence: alaam@mit.edu

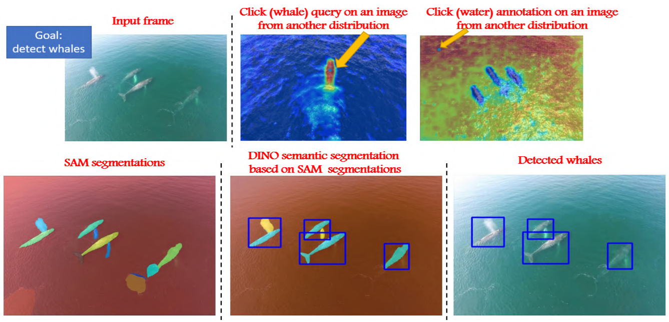

Goal: detect whales
Input frame
Click (whale) query on an image from another distribution
Click (water) annotation on an image from another distribution
SAM segmentations
DINO semantic segmentation based on SAM segmentations
Detected whales

Fig. 2: *FAn* outputs illustrations on input frame of 4 whales with a click query on a whale and a click query on water. First, SAM extracts multiple masks, then, based on DINO features, *FAn* classifies each mask to what object it refers from the given queries (water/whales). Finally, whales are detected by assigning the masks whose DINO feature descriptor is closest to the whales' query descriptor. NOTE: Heat maps are shown in the click (query) figures.

have shown impressive open-set interaction abilities [38]–[42], and extended robustly to multimodal applications [43]–[46]. However, integrating these models into real-time resource-constrained robotic systems poses significant challenges, due to their large model size and high inference latency.

A. Our Contributions

We address the pre-discussed gaps by developing an *open-set* real-time any object following approach, which can flexibly adapt to categories specified at inference time, via multiple modalities including text, images, and clicks. Specifically, we present the *follow anything* system (***FAn***):

- an **open-set**, **multimodal** approach to detect, segment, track, and follow *any* object in **real-time** ($>$ 6FPS on a 8GB GPU). The desired object may be specified via a text prompt, an image, a bounding box, or a click.
- a *unified* system that is easily deployed on a robot platform (in our work, a micro aerial vehicle). The system includes real-time processors for input image streams and visual-servoing control loops for following the object of interest.
- built with *re-detection mechanisms* that account for scenarios where the object of interest is occluded or tracking is lost. This mechanism can function autonomously or with human guidance, ensuring the object is successfully identified and tracked again, maintaining continuity in the tracking process.

We validate our system by autonomously detecting, tracking, and following a multitude of mobile agents including a drone, an RC car, and a manually operated brick.

II. Our Approach: *FAn*

Open-vocabulary object following: Given (1) a robotic system (here, a micro aerial vehicle) equipped with an onboard camera, and (2) an object of interest within the onboard camera's field-of-view (specified either as a text prompt, an image, a bounding box, or a click); the object following task involves detecting the object of interest, and producing robot controls u_t at each time step t such that the object of interest is constrained to always completely be within the field of view of the onboard camera. This is an extremely challenging task; it necessitates correctly identifying the object of interest and determining its position relative to the robot's onboard camera frame, all the while accounting for variations in the environment, background clutter, object size, etc. It also then requires the object to be continuously tracked across time; while at the same time, the robot controller needs to output a sequence of stable velocities (or accelerations) and simultaneously ensure the stability of the robot and the visibility of the tracked object.

***FAn* system overview**: *FAn* uses a combination of state-of-the-art ViT models, optimizes them for real-time performance, and unifies them into a single system. In particular, we leverage the segment anything model (SAM) [35] for segmentation, DINO [34], and CLIP [33] for general-purpose visual features, and design a lightweight detection and semantic segmentation scheme by combining the features from CLIP and DINO with the class-agnostic instance segmentation determined by SAM. We use the (Seg)AOT [47], [48] and SiamMask [49] models for real-time tracking, and design a lightweight visual servoing controller for object following.

A. Real-time open-vocabulary object detection

We first describe our lightweight object detection and segmentation pipeline that builds atop SAM, CLIP, and DINO. Our system takes as input an RGB frame from a video stream, represented by a 3D tensor $F \in \mathbb{R}^{h\times w\times 3}$, and a query q representing the desired object to detect in the video, (e.g., a text "*a blue whale*", an image of a whale, or a click on a whale from another image). The object detection subsystem is tasked to detect the object specified by the user query q in an input image frame. We use $\mathtt{Seg}$ to denote the class-agnostic instance segmentation operator (SAM [35] or Mask2Former [50]). $\mathtt{Seg}$ takes as an input the current frame F, and outputs a set of n masks $\{M_1,\cdots,M_n\} := \mathtt{Seg}(F)$ (n depends on the input frame and is not a constant), where each mask $M_i \in \mathbb{R}^{h\times w}$ is a binary matrix with ones in the indices of pixels defining the corresponding segmented object, and zeros elsewhere. We also use $\mathtt{Desc}$ to denote a feature extractor model; which in our case is either the DINO or CLIP vision transformer (ViT) model. These models extract pixel-wise feature descriptors using techniques described e.g., in [51], [52] and summarized in Section II-C. $\mathtt{Desc}$ receives as an input the current frame F, and outputs a descriptor tensor $D := \mathtt{Desc}(F) \in \mathbb{R}^{h\times w\times d}$, where for every pixel in F, a descriptor vector of dimension d is constructed. This d dimensional vector encapsulates the semantic information about its corresponding pixel. Additionally, $\mathtt{Desc}$ can be also used to provide a feature descriptor $v := \mathtt{Desc}(q) \in \mathbb{R}^d$ for the input query q.

Embedding input queries. To detect the desired object referred to by the query q, we start by computing the feature descriptor of the query q: $v := \mathtt{Desc}(q)$, such that v encodes the information in the feature space representing the object described by the query q. Now the system starts receiving frames from the stream, and for every frame F_i ($i := 1,2,3\cdots$), *FAn* applies the following steps.

First, we compute the (binary) instance segmentation masks by applying $\mathtt{Seg}$ on F_i, $\{M_1,\cdots,M_n\}_i := \mathtt{Seg}(F_i)$. Intuitively, this step partitions the frame into n objects (regions) and a

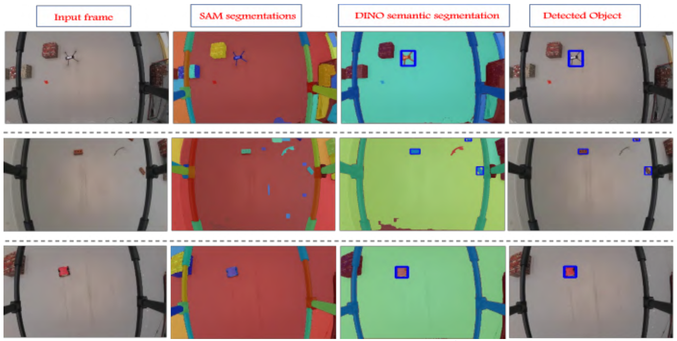


Fig. 3: **Automatic detection experiments (SAM-and-DINO).** Examples of our automatic detection scheme for detecting Drones, Bricks, and RC Cars. The examples include (from left to right): the original input frame, the outputs of SAM segmentation masks, and DINO+Cosine similarity semantic segmentation and detection.

background, however, none of these objects are classified as labeled/identified objects. Additionally, these regions might intersect. Hence, what is missing, is to predict for each region, whether it is the desired object to track or not. In the case where a set of queries $Q = \{q_1, \cdots, q_m\}$ is given, the goal is to classify which query amongst the m provided matches (if any) best each segment $M_j \in \texttt{Seg}(F_i)$. This brings us to the second step.

Second, we extract the pixel-wise descriptors by applying $\texttt{Desc}$ on F_i, $D_i := \texttt{Desc}(F_i) \in \mathbb{R}^{h \times w \times d}$. After this step, D_i contains $h \cdot w$ descriptors, where each descriptor corresponds to a pixel in the input image. To compare each region (mask) with the given input query q, we need to aggregate these per-pixel descriptors to form region-level descriptors. We find the average pooling aggregation operator to be fast and effective for this purpose. This not only provides us with a more generic descriptor encapsulating all of the features across the specific mask but also improves the performance of the downstream system modules. Opposed to comparing the q query's feature descriptor v to all of the per-pixel descriptors associated with a specific mask, we only need to compare the aggregate region-level descriptors. Thus, the next step in our pipeline involves computing the mean feature descriptor v_j for each segmentation region, i.e., for every $j \in \{1, \cdots, n\}$: $v_j := \frac{1}{\texttt{non-zero}(M_j)} \sum_{p \in D_i[M_j]} p$, where $\texttt{non-zero}(M)$ denotes the number of non-zero entries in binary matrix M, and $D_i[M_j]$ denotes the set of d dimensional vectors from D_i corresponding to the non-zero pixel entries in the mask M_j. The vector v_j, encodes the semantic information representing the region of the segment M_j in the features space. For every region (segment/mask) $j \in \{1, \cdots, n\}$, we have its corresponding descriptor v_j.

Similarity scores: Given a query (in the form of text, image, or click), we first extract a query feature descriptor v by applying a modality-specific encoder (CLIP for text-query, DINO or CLIP image encoder followed by average pooling for image-query, directly selecting the closest pixel/patch feature for click-query). To match this query to the current image, we compute the cosine similarity between each region descriptor v_j, and the query feature descriptor v as $\cos(v_j, v) := \frac{v_j^T v}{\|v_j\| \|v\| + \varepsilon}$, where $\varepsilon > 0$ is a small constant, for numerical stability. This is the fourth step, and it intuitively measures how similar each mask (region) is to the query features descriptor.

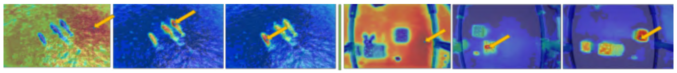


Fig. 4: Heat maps showing the pixels' semantic similarity. For every pixel, its feature descriptor is extracted then cosine similarity is computed between its descriptor and a focal point pixel descriptor (pointed at by a yellow arrow).

Single query detection. If the similarity $\cos(v_j, v)$ between the given query and the mask feature descriptor is larger than a given threshold α, we assign the region corresponding to this mask in the original frame the label of the query.

Multi-class detection. Should the user provide a set of queries $Q = \{q_1, \cdots, q_m\}$, the system computes the descriptor $v^k := \texttt{Desc}(q_k)$ for every $q_k \in Q$, then, for every pair of query descriptor v^k and region descriptor v_j, it computes: $\cos(v_j, v^k)$. Now, for every $j \in \{1, \cdots, n\}$, it finds its most similar query: $\max_{k \in \{1, \cdots, m\}} \cos(v_j, v^k)$. Finally, if the cosine similarity between the query vector (v^k), and the mask descriptor (v_j), exceeds a threshold α, we assign the label of the query to the region in the original frame corresponding to this mask, otherwise, it is considered "non-labeled".

After this process, each pixel is assigned a label from $\{1, \cdots, m\}$, or 0 if unlabeled. Figure 2 provides an illustration of the whole detection flow, and Figures 3 and 10 present results on detecting objects via SAM+DINO, and SAM+CLIP respectively.

Manual queries: We provide the users an option to manually draw bounding boxes (or provide outputs from a customized domain-specific detector) around the objects they wish to track, or alternatively, click on one or two pixels within the object (in real-time from the video stream). After user selection, we use SAM to accurately segment and obtain the object mask. This method ensures precise control over tracking, making it suitable for high-accuracy detection scenarios.

B. *Fast detection for limited hardware*

Off-the-shelf implementations of foundation models like SAM and DINO are not well-suited for real-time onboard detection, segmentation, and tracking. SAM takes several seconds to compute segmentation masks per frame. While we evaluated the recently proposed FastSAM [53] model and obtained a 15× speedup on our hardware with comparable performance, the best runtime achieved by FastSAM is between 10 and 12 FPS, which is still insufficient for detecting fast-moving objects. This is because segmentation outputs also need to be supplemented by features from ViT models, and the detection submodules.

Fast detection by (solely) grouping DINO features: To mitigate this compute bottleneck, we instead propose to first obtain coarse detections by grouping DINO features. These coarse detections may further be refined by periodically computing segmentation masks and tracking these over time, effectively rendering the overall system operable at high frame rates. To obtain coarse detections, (i) we extract the pixel-wise descriptors by applying $\texttt{Desc}$ (DINO) on the current input frame F_i, $D_i := \texttt{Desc}(F_i) \in \mathbb{R}^{h \times w \times d}$, (ii) given the inputs set of queries $Q = \{q_1, \cdots, q_m\}$, the system computes the

TABLE I: Runtime in frames per second (FPS) for all of the used models on an NVIDIA GeForce RTX 2070 onboard a laptop.

Model	FPS frame size 320 × 240	FPS frame size 640 × 480	Subtask
SAM (points_per_side = 16)	0.71	0.58	
SAM 16BIT (points_per_side = 16)	0.97	0.71	Segmentation
FASTSAM	10.7	10.2	
DINO	4.76	4.68	
DINO TRACED	6.27	NA	Feature extraction
DINO 16BIT	10.63	11.55	for click/image queries
DINO 16BIT+TRACED	17.46	17.44	
CLIP	7.81	7.65	Feature extraction
CLIP 16BIT	21.12	20.21	for text queries
SIAMMASK	50.3	49.2	Tracking
DEAOT	28.74	17.13	

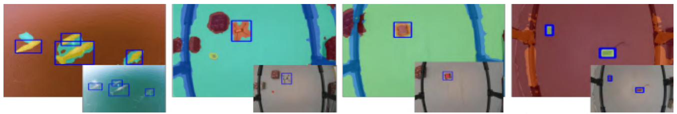


Fig. 5: **Fast automatic detection experiments (DINO only):** Examples of our fast automatic detection scheme on detecting (1) whales, (2) drones, (3) RC cars, and (4) toy bricks. This approach is much faster and works very well for detecting the desired object. However, it provides a less ”clean” segmentations/masks.

cosine similarity $\cos(v_{h,w}, v^k)$ for each pair of query $q_k \in Q$ (where $v^k := \texttt{Desc}(q_k)$) and pixel-wise descriptor vector $v_{h,w}$. Next, as previously, (iii) for each pixel, it picks the closest (most similar) query, i.e., the one with the maximum cosine similarity. Now, (iv) if the cosine similarity between the query vector v^k and the pixel feature descriptor $v_{h,w}$ surpasses a specified threshold α, we assign the label of the query to the corresponding pixel in the original frame F_i. Otherwise, it is considered as ”non-labeled”. Then, (v) we build a binary matrix $B_i \in \mathbb{R}^{h \times w}$ with 1 in pixels that are mapped to the desired object (to detect) and 0 elsewhere. Finally (vi) apply the `cv2.connectedComponents` function on B_i. This function receives a binary image (B_i) where white regions (pixels with label 1) on a black background (pixels with label 0) represent connected components. The function assigns unique integer labels to each connected component and labels background pixels as 0. We have used it since we might detect more than one object, each in a different region of the frame, this function provides us with each object with its unique mask. See Figure 5 for experiments leveraging the detection module proposed here.

Optimizing DINO runtime: We speed up DINO using two optimization techniques: Quantization (reduces numerical precision) and tracing (converts dynamic graphs into static ones). See Table I for runtime details of all the used models in our system. We report the running time for each model independently, not as part of the whole system. Note that some models automatically reshape inputs to a constant size. We also compare the runtime of our detection phase, with the popular Grounded-SAM [54] method in Table II.

C. Extracting per-pixel feature descriptors

While a few methods adapt foundation models like CLIP to provide per-pixel descriptors, these methods [55]–[58] require model re-training or finetuning on an image-text aligned dataset. This often results in concepts absent in the fine-tuning set being forgotten by the models as demonstrated in ConceptFusion [52]. To counteract this, [52] presents a zero-shot method for constructing pixel-aligned features that combine local (region-level) data with global (image-level) context included in models like CLIP. For efficiency (real-time processing) purposes, we adapt part of this method in our system when using CLIP for providing pixel-wise feature descriptors, however, we only use their ablated baseline which computes purely local 2D features by extracting a bounding box around each segmentation mask (obtained from SAM) and passes them through the CLIP encoder. For DINO, we use [51] as is, and find that their pixel-wise feature descriptors are inherently informative and more efficient.

TABLE II: Runtime in frames per second (FPS) for the detection phase of the system on an NVIDIA GeForce RTX 2070, compared to the popular open-source library [54], using a more powerful GPU of NVIDIA RTX 3090.

Approach	FPS frame size 320 × 240	FPS frame size 640 × 480
OURS WITH SAM	0.601	0.536
OURS WITH FASTSAM	9.153	9.172
OURS WITH JUST DINO (NO SAM OR FASTSAM)	15.67	14.37
GROUNDED-SAM [54]	0.508	0.51

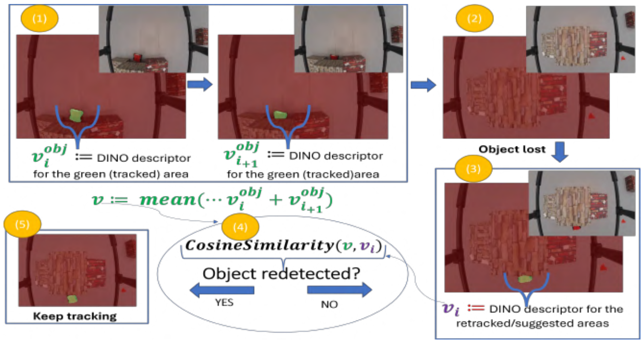
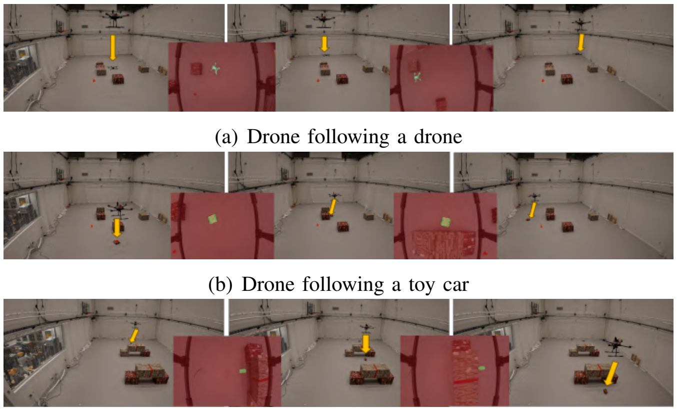

(1)
v_i^{obj} := DINO descriptor for the green (tracked) area
v_{i+1}^{obj} := DINO descriptor for the green (tracked)area
(2)
Object lost
(3)
v_i := DINO descriptor for the retracked/suggested areas
$v := mean(\cdots v_i^{obj} + v_{i+1}^{obj})$
(4)
$CosineSimilarity(v, v_i)$
Object redetected?
YES
NO
(5)
Keep tracking

Fig. 6: **Automatic re-detection via cross trajectory stored ViT features.** (1) At every frame, we store the DINO features representing the tracked object. (2) Once the object is lost, we (3) either apply a segmentation model or get suggested masks from the tracker, for every mask, we compute the DINO descriptors, and (4) compare it to the pre-computed ones. If a high similarity is obtained we keep tracking the object, else, we repeat (3) on the next frame.

D. Re-detecting a lost object

We offer three re-detection methods for temporary object loss during tracking, catering to different needs. Our system automatically initiates re-detection when needed, and users can choose the level of support before starting the *FAn* pipeline: The first level relies on the tracker to re-detect the object, it's the fastest and less robust, occasionally leading to false detections of similar objects. The second approach involves human-in-the-loop re-detection, requiring a user to click/draw a bounding box when tracking is lost, assuming human availability, which isn't always possible. To mitigate this, we also propose an automatic re-detection technique.

Automatic re-detection via cross trajectory stored ViT features. To enable a robust and accurate autonomous re-detection of the tracked (lost) object, we provide a feature-descriptor storing mechanism for the tracked object in different stages of the tracking process, these stored features, will be used to find the object once lost. Specifically, we suggest the following. Let $\tau > 0$ be an integer. During the tracking, at each iteration i such that $i \bmod \tau = 0$, define M_i^{obj} to be the mask denoting the current tracked object in the frame, we first apply `Desc` on the current frame F_i to obtain $D_i := \texttt{Desc}(F_i) \in \mathbb{R}^{h \times w \times d}$, then we compute the mean descriptor of the current tracked object as: $v_i^{obj} := \frac{1}{\texttt{non-zero}(M_i^{obj})} \sum_{p \in D_i[M_i^{obj}]} p$. This feature represents the tracked object in the ith step. We thus store this descriptor



(a) Drone following a drone

(b) Drone following a toy car

(c) Drone following a toy (manually moved) brick

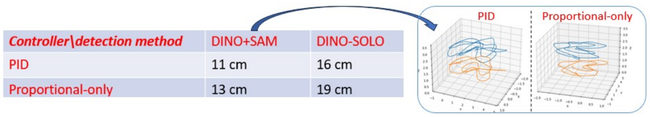
Fig. 7: **Automatic tracking, following, and re-detection.** The tracked object is referred to by the yellow arrow, we also show the results of the re-detection mechanism in the last two rows.

and add it to the set of previously computed descriptors to obtain the set $V^{obj} := \left\{v_0^{obj}, v_\tau^{obj}, v_{2\tau}^{obj}, \cdots, v_i^{obj}\right\}$. Now, whenever the system loses the tracked object, we apply the following recovery mechanism. The system goes back to the detection stage, with a query feature descriptor at hand as $v = \frac{1}{\left|V^{obj}\right|}\sum_{v^{obj}\in V^{obj}} v^{obj}$, seeking the closest region from the segmented frame, and thus re-detecting the object. Here, the segmentation might be given by the segmentation model (e.g., SAM), or by the tracker which tries to re-detect the lost object. Note we use the mean to gain faster performance for real-time applications, however, other techniques can be used to improve the robustness; see Figure 6.

III. Experiments

Hardware. We use a quadrotor equipped with an RGB camera (see Figure 9). The quadrotor is custom-built with a Pixhawk running PX4 flight control software. The camera data is streamed directly to a remote ground station computer equipped with an NVIDIA GeForce RTX 2070, and Intel i7-10750H CPU, with Ubuntu 20.04.5 LTS, using the "herelink" digital transmission system along with other telemetry data. The ground station runs the tracking algorithm and sends control commands to the quadrotor via Mavlink. To enable indoor testing, the quadrotor is also equipped with an onboard computer that runs MAVROS and interfaces with an external Vicon motion capture system to get the position.

Implementation and system details. We outline key details of our system: **Run-time improvement.** We enhance segmentation/detection performance by compressing SAM and DINO through quantization and tracing and using FastSAM. For tracking, we offer support for the fast SiamMask [49] tracker; see Table I for runtime (FPS) details. **Flight controller.** For versatility, we used PX4, open-source flight control software, to interface with our quadrotor. The MAVSDK Python library is used to send velocity commands for 3D motion and yaw control, streamlining integration with PX4-based drones in future projects. **Visual servoing.** We mount the onboard camera on the bottom of the quadrotor facing the ground. At relatively small translational velocities the first-order approximations of roll and pitch angles are close to zero. In addition, we fixed the drone altitude and yaw angle. This simplifies the visual servoing task to 2D plane tracking using proportional control. We use a proportional controller based on pixel distances to center the object in the frame and employ a lowpass filter to smooth quadrotor trajectories, ensuring accuracy in challenging scenes. **Video Streaming.** To process frames from an online video stream in real-time, we implemented a low-latency online streamer using the OpenCV library in Python. This streamer continuously reads frames with a parallel thread and maintains a buffer size of 1, ensuring immediate access to the latest frame when needed. **Software.** We mainly use Torch, cv2, and mavsdk; see our project page for full details.



Controller\detection method	DINO+SAM	DINO-SOLO
PID	11 cm	16 cm
Proportional-only	13 cm	19 cm

Fig. 8: **Trajectory comparison.** We report the mean Euclidean distance between every point in the x, y plane of the quadrotor and its aligned point in the plane (closest point) of the followed object.

A. Real time object following exprements

We tested (i) our overall system for detecting, tracking, re-detecting, and following: RC cars, drones, and bricks in real-time. Here we used SAM+DINO and DINO-SOLO approaches for the detection task on all of the tested objects - the provided queries are clicks on the desired objects from other pictures (we provide a script for obtaining these click queries). Both approaches worked seamlessly for detecting and tracking the desired objects. (ii) We demonstrate our system **for re-detecting** an object that gets occluded from the scene during tracking. Specifically, during the following experiments, the RC car and the brick pass under a "tunnel" twice, and our re-detection mechanism is able to recover and resume tracking. Figures 7(a), 7(b), and 7(c) show different scenarios during the following. We encourage the reader to view the demos on our project webpage and in the explainer video. (iii) In addition, we recorded the actual **3D trajectory** coordinates of the following quadrotor and the target object to assess the robustness of our tracking system. Specifically, we recorded continuous tracking data for over 4 minutes while following a ground robot. We report the mean Euclidean distance between every point in the x, y plane of the quadrotor and its aligned point in the plane (closest point) of the followed object. This experiment was conducted 4 times; using PID vs proportional-only as a controller, and using SAM+DINO vs DINO-SOLO as a detector. The results are reported in Figure 8. We can see that the drone follows the object smoothly and accurately using the different controllers and detectors. We also visualize both trajectories for the case of SAM+DINO.

B. Zero-shot detection exprements

Data. We stored the tracking and detection streams from the SAM+DINO following experiment and used it to test the *FAn* system and its different variants for zero-shot detection. For each of the tested objects, we picked multiple frames during the tracking and detection showcasing diverse object positions and diverse scenes. Other than that, we also use our private set of whale images to test on.

TABLE III: true positive detections divided by the number of object appearances (provided next to the object name), and number of false positive detections (number in brackets if any); single query test.

Approach	**Car** (11)	**Drone** (15)	**Bricks** (15)	**Whales** (25)
SAM+DINO	1.0	0.5	0.6	0.84 (2)
SAM+CLIP	0.81 (3)	0.4 (1)	NA	0.8 (1)
DINO-SOLO	0.91	0.7	0.73	0.92 (1)
10-MEANS	1.0	0.5	0.6	0.84 (3)
5-MEANS	0.91	0.4	0.53	0.8 (2)
MAJORITY VOTING	1.0	0.5	0.53	0.84 (3)

TABLE IV: true positive detections divided by the number of object appearances (provided next to the object name), and number of false positive detections (number in brackets if any); multiple queries test.

Approach	**Car** (11)	**Drone** (10)	**Bricks** (15)	**Whales** (25)
SAM+DINO	1.0	0.6	0.67	0.84
SAM+CLIP	0.91 (2)	0.5 (1)	0.4 (5)	0.65
DINO-SOLO	1.0	0.9	0.87	0.92 (1)
10-MEANS	1.0	0.6	0.67	0.8
5-MEANS	1.0	0.5	0.6	0.8
MAJORITY VOTING	1.0	0.6	0.67	0.84

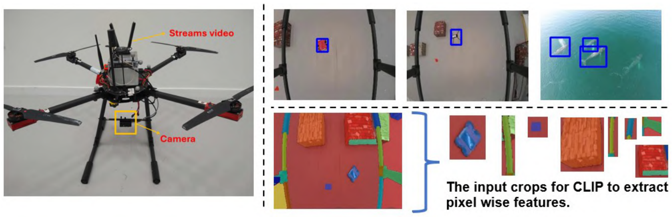


Fig. 9: **Left:** Our custom-built quadrotor. **Right-up:** Successful automatic detection via text queries (SAM+CLIP) on low-resolution images; text queries used from left to right: ”a toy car” (single query), ”a drone” (single query), and ”a whale”+”water” (multiclass). **Right bottom:** In some cases, the raw data from the cropped masks (to get pixel-wise features from CLIP) does not provide enough information for CLIP - since the image is of low resolution and the mask is small causing it to provide not accurate descriptors and thus *FAn* may not detect the objects.

Comparison. We quantitatively compare the suggested methods and analyze their advantages and disadvantages. We applied each of SAM+DINO, SAM+CLIP, and DINO-SOLO to assess their efficacy in detecting the object within the given data. We report both True Positive and False Positive detection results. Furthermore, we conducted a comparative analysis involving an alternative version of our approach, which consists of two variations. (i) Majority Voting: We assigned each pixel in the mask to its most similar query, and subsequently assigned the mask the label that was most frequently selected across all mask pixels. (ii) K-Means: For each mask, we retained a set of $K > 1$ representatives based on the K-means algorithm. We then gauged the similarity of these representatives with the provided queries and assigned the mask a label based on the majority consensus among these K representatives. Our testing encompassed two scenarios: (i) The system was presented with multiple queries representing the environment, including ”a robot leg, a box, a ground” (in the whales experiment, these queries were replaced with ”water”), along with the desired query ”a drone, a toy car, a brick, a whale” (Table IV) and (ii) The system was given a single desired query (Table III).

The threshold α. For all methods, we tuned α to minimize the false positive detections while achieving a fine true positive detection rate; In our system, it’s acceptable to not immediately identify the intended object, but our priority is to prevent the detection and tracking of an incorrect target. We used 0.35 for SAM+DINO and 0.23 for DINO+CLP in all experiments. For DINO-SOLO 0.4 and 0.6 were used in the multiple queries and single query experiments, respectively.

C. Mask quality experements

We compare the mask quality of our detection methods (DINO-SOLO, SAM+DINO/CLIP). We use the first video from the Cholec80 dataset [59], which has mask annotation for body parts and tools across frames during surgery. We aimed to detect the “grasper” tool and track it across frames. Table V reports (1) the mean intersection over union (mIoU) of the detection and the annotated data across frames and (2) the true positive detection percentage of the desired object, we also test how the detected mask quality affects the tracking; we report (3) the mIoU of the desired tracked object after each of the detection methods detected it. Queries: ”body part”, ”background”, and ”surgery tool”.

D. Discussion and conclusions

SAM+DINO. Figures 3 and 2 show example results for real-time detection via SAM+DINO. Tables IV and III, indicate that the detection achieves a high level of accuracy for cars and whales, and performs well for drones and bricks - but may occasionally miss certain instances. After analyzing the results, it becomes apparent that when SAM generates reliable regions/segmentation, DINO consistently assigns the correct labels to each of these regions, ensuring precise and appropriate object detection. However, in cases where SAM fails to capture these regions accurately (resulting in inadequate segmentations), the object goes undetected. This scenario is exemplified by 4 drone object in the dataset and 3 bricks, where SAM fails to identify the mask of the drone/brick (see Figure 11). Regarding accurate DINO classifications, we offer explanations illustrated in Figure 4. These figures depict heatmaps based on cosine similarity calculations between DINO feature descriptors of each pixel and a designated focal point pixel. The visualizations demonstrate that pixels sharing similar semantic characteristics exhibit a high degree of similarity in their DINO features.

DINO-SOLO. In Figure 5 we show several examples showcasing the efficiency of our rapid automated detection system. This approach is significantly faster and performs admirably in detecting the desired objects. Even more, in many cases when SAM misses providing the desired object a mask, using DINO-SOLO can still detect the object. However, the resulting masks are not of high quality compared to the masks obtained from SAM, and this may potentially affect the tracking performance; see Figure 11 and Table V.

SAM+CLIP. Examples for detection via ”text” prompts through SAM+CLIP are shown in Figure 9. For the tested low-resolution images, SAM+CLIP detections are not as robust, the method yields less precise similarity scores, increasing the likelihood of missed detections, particularly for objects lacking unique shapes like the brick. Additionally, in some cases, as the image has low resolution, if the object has a small (correct) mask, it does not present enough raw information and is thus misclassified. Figure 9 shows an example of low-resolution images for such scenarios; we further discuss why this happens when using CLIP and not DINO in our discussion later. We note that this method is still beneficial for our

TABLE V: mIoU for tracking and detection (DINO-SOLO vs SAM+DINO or CLIP). We also report the accuracy of the detection (percentage of true positive detections from all of the parsed frames). Note that in the tracking SAM+DINO and SAM+CLIP got the same results as they were provided the same mask by SAM (the first detected). The used tracker is DEAOT.

Approach	Stage	mIoU (480x854)	mIoU (240x427)	Acc (480x854)	Acc (240x427)
SAM+DINO	Tracking	0.87	0.77	NA	NA
SAM+CLIP	Tracking	0.87	0.77	NA	NA
DINO-SOLO	Tracking	0.84	0.75	NA	NA
SAM+DINO	Detection	0.86	0.81	0.89	0.89
SAM+CLIP	Detection	0.82	0.68	0.18	0.13
DINO-SOLO	Detection	0.73	0.67	0.98	0.98

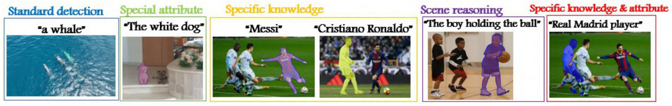


Fig. 10: Automatic detection experiments via text queries (SAM-and-Clip) on high resolution data.

system, the main idea is that we need one accurate detection with high confidence (e.g., with further increasing α) for the desired object and then we can start the object-following scheme, thus, we can still benefit from the multimodality of the system. Additionally, as this method requires only the text prompt and not an image/clicks, it is much easier to utilize. To verify our claims regarding the reason for the dropped performance of *FAn* when using SAM+CLIP, we tested it on high-resolution images. Here, the reasoning and detection are robust justifying our claims. We conduct the following 4 tests: (i) Standard detection, e.g., "detect a whale", (ii) scene reasoning-based detection, e.g., "detect the boy *holding the ball*". (ii) Special attribute-based detection, like, "detect the *white* dog". (iv) Special prior knowledge-based detection. In this case, the system should have prior knowledge of a specific object like its name/nickname. For example "detect Messi/Cristiano Ronaldo". (v) Special prior knowledge& attribute based detection. e.g., "detect a Real Madrid player". See result in Figure 10.

SAM limitations: With vs without. SAM might miss important regions in the image. When the desired object is in these regions it will be impossible to detect it and thus DINO+SAM yields fewer true positive detections compared to DINO-SOLO. On the other hand, DINO+SAM provides high-quality masks once the object is detected while DINO-SOLO masks are less refined. See Tables V, III, and IV.

Queries. Using **multiple** queries to annotate other objects that might be in the scene reduces the number of False positives leading to a more robust and reliable system.

DINO vs CLIP. The method we are using to obtain pixel-wise features from DINO [51] is faster and provides better descriptors for every pixel compared to the method used for CLIP. This is because it requires one forward pass on the whole image to compute the per-pixel features. In addition, when using DINO, the method computes the per patch/pixel features while taking into count the full image, as it simply utilizes the patch-wise descriptors (outputs of the query, key, or value matrix in some attention layer of the transformer) of DINO, thus providing descriptors with richer context of the whole image. In CLIP, the method uses SAM to extract masks [52] and then applies CLIP on crops of these masks to extract features for all pixels in this mask, thus, it is less efficient and might yield less meaningful features when applying CLIP on small crops with limited raw data.

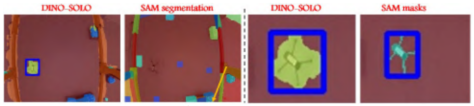


Fig. 11: **With vs without SAM.** Right: SAM creates high-quality segmentation masks compared to DINO-SOLO (not using SAM). Left: SAM might miss important regions in the image.

The competing methods. We found no improvement with other variants like K-means and majority voting; often, our original methods performed better. Also, the K-means variant runs at 0.03 FPS, and the majority voting runs at 0.32 FPS.

Summary. *FAn* bridges the gap between SOTA computer vision and robotic systems, providing an end-to-end solution for detecting, tracking, and following any object. Its open set, multimodal, real-time capabilities, adaptability to different environments, and open-source code make it a valuable tool.

References

[1] B. Taha and A. Shoufan, "Machine learning-based drone detection and classification: State-of-the-art in research," *IEEE Access*, vol. 7, pp. 138 669–138 682, 2019. 1

[2] A. Maalouf, Y. Gurfinkel, B. Diker, O. Gal, D. Rus, and D. Feldman, "Deep learning on home drone: Searching for the optimal architecture," *arXiv preprint arXiv:2209.11064*, 2022. 1

[3] H. Naeem, J. Ahmad, and M. Tayyab, "Real-time object detection and tracking," *INMIC*, pp. 148–153, 2013. 1

[4] A. Koubâa and B. Qureshi, "Dronetrack: Cloud-based real-time object tracking using unmanned aerial vehicles over the internet," *IEEE Access*, vol. 6, pp. 13 810–13 824, 2018. 1

[5] A. Restas *et al.*, "Drone applications for supporting disaster management," *World Journal of Engineering and Technology*, vol. 3, no. 03, p. 316, 2015. 1

[6] D. Tezza and M. Andujar, "The state-of-the-art of human–drone interaction: A survey," *IEEE Access*, vol. 7, pp. 167 438–167 454, 2019. 1

[7] H. K. Cheng, Y.-W. Tai, and C.-K. Tang, "Modular interactive video object segmentation: Interaction-to-mask, propagation and difference-aware fusion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 5559–5568. 1

[8] X. Lu, W. Wang, M. Danelljan, T. Zhou, J. Shen, and L. Van Gool, "Video object segmentation with episodic graph memory networks," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer, 2020, pp. 661–679. 1

[9] Z. Yang, J. Miao, X. Wang, Y. Wei, and Y. Yang, "Scalable multi-object identification for video object segmentation," *arXiv preprint arXiv:2203.11442*, 2022. 1

[10] Z. Yang, Y. Wei, and Y. Yang, "Associating objects with transformers for video object segmentation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 2491–2502, 2021. 1

[11] Z. Yang and Y. Yang, "Decoupling features in hierarchical propagation for video object segmentation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 36 324–36 336, 2022. 1

[12] S. Cho, M. Lee, S. Lee, C. Park, D. Kim, and S. Lee, "Treating motion as option to reduce motion dependency in unsupervised video object segmentation," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 5140–5149. 1

[13] X. Lu, W. Wang, C. Ma, J. Shen, L. Shao, and F. Porikli, "See more, know more: Unsupervised video object segmentation with co-attention siamese networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 3623–3632. 1

[14] W. Wang, X. Lu, J. Shen, D. J. Crandall, and L. Shao, "Zero-shot video object segmentation via attentive graph neural networks," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 9236–9245. 1

[15] W. Wang, H. Song, S. Zhao, J. Shen, S. Zhao, S. C. Hoi, and H. Ling, "Learning unsupervised video object segmentation through visual attention," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 3064–3074. 1

[16] Y. Cheng, L. Li, Y. Xu, X. Li, Z. Yang, W. Wang, and Y. Yang, "Segment and track anything," *arXiv preprint arXiv:2305.06558*, 2023. 1

[17] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," *Advances in neural information processing systems*, vol. 29, 2016. 1

[18] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 781–10 790. 1

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587. 1

[20] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448. 1

[21] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015. 1

[22] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788. 1

[23] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020. 1

[24] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018. 1

[25] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 2016, pp. 21–37. 1

[26] H. Xing, Y. Wang, X. Wei, H. Tang, S. Gao, and W. Zhang, "Go closer to see better: Camouflaged object detection via object area amplification and figure-ground conversion," *IEEE Transactions on Circuits and Systems for Video Technology*, 2023. 1

[27] P. Nousi, I. Mademlis, I. Karakostas, A. Tefas, and I. Pitas, "Embedded uav real-time visual object detection and tracking," in *2019 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, 2019, pp. 708–713. 1

[28] J. Zhao, J. Zhang, D. Li, and D. Wang, "Vision-based anti-uav detection and tracking," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 12, pp. 25 323–25 334, 2022. 1

[29] S. R. Ganti and Y. Kim, "Implementation of detection and tracking mechanism for small uas," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2016, pp. 1254–1260. 1

[30] E. Unlu, E. Zenou, N. Riviere, and P.-E. Dupouy, "Deep learning-based strategies for the detection and tracking of drones using several cameras," *IPSJ Transactions on Computer Vision and Applications*, vol. 11, no. 1, pp. 1–13, 2019. 1

[31] R. Barták and A. Vykovský, "Any object tracking and following by a flying drone," in *2015 Fourteenth Mexican International Conference on Artificial Intelligence (MICAI)*, 2015, pp. 35–41. 1

[32] A. Barisic, M. Car, and S. Bogdan, "Vision-based system for a real-time detection and following of uav," in *2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS)*, 2019, pp. 156–159. 1

[33] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763. 1, 2

[34] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 9650–9660. 1, 2

[35] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo *et al.*, "Segment anything," *arXiv preprint arXiv:2304.02643*, 2023. 1, 2

[36] C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. Le, Y.-H. Sung, Z. Li, and T. Duerig, "Scaling up visual and vision-language representation learning with noisy text supervision," in *International conference on machine learning*. PMLR, 2021, pp. 4904–4916. 1

[37] A. Guzhov, F. Raue, J. Hees, and A. Dengel, "Audioclip: Extending clip to image, text and audio," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 976–980. 1

[38] S. Tellex, N. Gopalan, H. Kress-Gazit, and C. Matuszek, "Robots that use language," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 25–55, 2020. 2

[39] Y. Bisk, A. Holtzman, J. Thomason, J. Andreas, Y. Bengio, J. Chai, M. Lapata, A. Lazaridou, J. May, A. Nisnevich *et al.*, "Experience grounds language," *arXiv preprint arXiv:2004.10151*, 2020. 2

[40] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022. 2

[41] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu *et al.*, "Rt-1: Robotics transformer for real-world control at scale," *arXiv preprint arXiv:2212.06817*, 2022. 2

[42] S. Li, X. Puig, C. Paxton, Y. Du, C. Wang, L. Fan, T. Chen, D.-A. Huang, E. Akyürek, A. Anandkumar *et al.*, "Pre-trained language models for interactive decision-making," *Advances in Neural Information Processing Systems*, vol. 35, pp. 31 199–31 212, 2022. 2

[43] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," in *International Conference on Machine Learning*. PMLR, 2021, pp. 8821–8831. 2

[44] K. Crowson, S. Biderman, D. Kornis, D. Stander, E. Hallahan, L. Castricato, and E. Raff, "Vqgan-clip: Open domain image generation and editing with natural language guidance," in *European Conference on Computer Vision*. Springer, 2022, pp. 88–105. 2

[45] O. Patashnik, Z. Wu, E. Shechtman, D. Cohen-Or, and D. Lischinski, "Styleclip: Text-driven manipulation of stylegan imagery," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 2085–2094. 2

[46] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *arXiv preprint arXiv:2204.06125*, 2022. 2

[47] Z. Yang and Y. Yang, "Decoupling features in hierarchical propagation for video object segmentation," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. 2

[48] Z. Yang, J. Miao, X. Wang, Y. Wei, and Y. Yang, "Scalable multi-object identification for video object segmentation," *arXiv preprint arXiv:2203.11442*, 2022. 2

[49] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, and P. H. Torr, "Fast online object tracking and segmentation: A unifying approach," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019. 2, 5

[50] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar, "Masked-attention mask transformer for universal image segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 1290–1299. 2

[51] S. Amir, Y. Gandelsman, S. Bagon, and T. Dekel, "Deep vit features as dense visual descriptors," *arXiv preprint arXiv:2112.05814*, 2021. 2, 4, 7

[52] K. M. Jatavallabhula, A. Kuwajerwala, Q. Gu, M. Omama, T. Chen, S. Li, G. Iyer, S. Saryazdi, N. Keetha, A. Tewari *et al.*, "Conceptfusion: Open-set multimodal 3d mapping," *arXiv preprint arXiv:2302.07241*, 2023. 2, 4, 7

[53] X. Zhao, W. Ding, Y. An, Y. Du, T. Yu, M. Li, M. Tang, and J. Wang, "Fast segment anything," *arXiv preprint arXiv:2306.12156*, 2023. 3

[54] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C. Li, J. Yang, H. Su, J. Zhu *et al.*, "Grounding dino: Marrying dino with grounded pre-training for open-set object detection," *arXiv preprint arXiv:2303.05499*, 2023. 4

[55] B. Li, K. Q. Weinberger, S. Belongie, V. Koltun, and R. Ranftl, "Language-driven semantic segmentation," *arXiv preprint arXiv:2201.03546*, 2022. 4

[56] H. Zhao, X. Puig, B. Zhou, S. Fidler, and A. Torralba, "Open vocabulary scene parsing," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2002–2010. 4

[57] G. Ghiasi, X. Gu, Y. Cui, and T.-Y. Lin, "Scaling open-vocabulary image segmentation with image-level labels," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXVI*. Springer, 2022, pp. 540–557. 4

[58] Y. Zhong, J. Yang, P. Zhang, C. Li, N. Codella, L. H. Li, L. Zhou, X. Dai, L. Yuan, Y. Li *et al.*, "Regionclip: Region-based language-image pretraining," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 16 793–16 803. 4

[59] W.-Y. Hong, C.-L. Kao, Y.-H. Kuo, J.-R. Wang, W.-L. Chang, and C.-S. Shih, "Cholecseg8k: a semantic segmentation dataset for laparoscopic cholecystectomy based on cholec80," *arXiv preprint arXiv:2012.12453*, 2020. 6