UAV Object-Based Semi-Autonomous and Autonomous Navigation

by

Amirmasoud Ghasemi Toudeshki

M.Sc., University of Tehran, 2014
B.Sc. University of Tehran, 2011

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

Declaration of Committee

Name:	**Amirmasoud Ghasemi Toudeshki**
Degree:	**Master of Science**
Thesis title:	**UAV Object-Based Semi-Autonomous and Autonomous Navigation**
Committee:	**Chair:** Alaa Alameldeen Associate Professor, Computing Science
	Richard T. Vaughan Supervisor Professor, Computing Science
	Angelica Lim Committee Member Assistant Professor, Computing Science
	Mo Chen Examiner Assistant Professor, Computing Science

Abstract

In this thesis, we have developed a semi-autonomous behavior that allows us to control a drone with less effort. We have also presented a technique that enables autonomous repeating of a previously traversed route using the visual navigation system. Our first application demonstrates an experiment with driving a drone using a vision-based control (visual servoing) method, particularly by tracking selected targets in an image view. In the second application, a drone equipped with a monocular camera has been derived manually on a path. Invariant semantic features (i.e., objects) have been extracted using an object detection neural network, YOLO. Using these features, we show that the drone can repeat the traversed route autonomously independently from the lighting condition and even appearance changes.

Keywords: Human drone interaction; visual teach and repeat, semantic features; navigation; computer vision

Acknowledgements

I want to thank Prof. Richard Vaughan, my senior supervisor, for his support, guidance, and helpful suggestions during my study and completion of my thesis. I am also very grateful to my supervisor Dr. Angelica Lim and my examiner, Dr. Mo Chen for their time and valuable feedback on this thesis. I would like to thank my lab members of the Autonomy Lab, especially Faraz Shamshirdar and Sepehr MohaimenianPour, for their support and for sharing their experiences and ideas with me. Finally, I am very thankful to my family, my mother, and my father for their endless support and passion, and to my love, Saeedeh, who gives me all her inspiration and encouragement I need in every step of my life.

Table of Contents

List of Figures

Chapter 1

Introduction

Nowadays, we see drones in many different applications, from delivery and inspection to fire-fighting and rescue. Early applications mainly were photography and video capturing, but due to the variety of new uses, perception and navigation have become essential components of drone intelligence. As drone usage increases, human drone interaction and autonomous drone navigation become more interesting for researchers [43]. The current state-of-the-art research in human-drone interaction includes developing and evaluating new control modalities and new human-drone communication methods. The control system consists of low-level layers to maintain the stability and commanding motor drivers directly and high-level layers for planning and navigation of high-level tasks such as autonomous and semi-autonomous actions. Data from inertial sensors such as gyroscopes and accelerometers is fed to low-level flight controllers to have stable hovering and smooth flight. Data from perception sensors like lidars and cameras is usually the input to high-level control and navigation layer and provides the required information for local and global mapping and localization.

For autonomous (or semi-autonomous) navigation of drones in an environment, sensory data translated to meaningful information can be combined with any prior knowledge about the environment to estimate the drone's location and map the field. In the robotic research area, Simultaneous Localization and Mapping (SLAM) of a robotic system is a popular topic which tries to construct or update a map of an unknown environment while simultaneously keeping track of a robot's location within it.

Sensors play an essential role in the accuracy of navigation systems. In contrast to fixed reference sensors like GPS with limited coverage and cumulative drift and divergence, cameras are suitable sensors for localization and navigation. Our focus in this thesis is on indoors and outdoors visual-only control and navigation methods for drones.

Vision-based control and navigation algorithms rely on visual features of the environment for the localization of drones. Prior to deep neural networks, handcrafted features such as SURF [1], SIFT [20], and ORB [36] were most widely used for navigation and localization tasks. Memory size is a challenge for the long-range autonomy of small devices such as drones, and since these features are very memory intensive, they are not suitable for real-world long-range applications. Also, these low-level features are not robust to lighting and viewpoint variation. In contrast, deep neural networks, particularly Convolutional Neural Networks (CNN) result in the semantic representation of an environment that can be used as robust high-level features to detect and recognize objects and activities.

In this thesis, we rely on the environment's visual features in our first work to semi-autonomously control a drone and, in our second work, to repeat a traversed route autonomously. Controlling a robot manually usually needs training and attention. We have developed a system that users can control a drone by clicking on the target point in a drones' camera feed using a touch-screen device. We have published this work in a poster presented at the IEEE/RSJ 30th International Conference of Intelligent Robots and Systems (IROS) [30], and we have extended the work in the thesis. Our second application exploits visual features to develop a visual-only route following system based on visual teach and repeat (VTR) methods. VTR has many applications, including but not limited to patrolling, inspection, delivering, and transportation. VTR was first presented in [10] in which a stereo camera was used to perform visual navigation for a ground robot. We apply this method to a flying drone with the significant difference in selecting visual features where we utilize semantic features as conditions and viewpoint-invariant features. On top of our visual teach and repeat method, We proposed a new motion controller that plans ahead to respond to upcoming objects and relocalizes the drone using sequential-temporal constraints. This work is published at the 15th Conference on Computer and Robot Vision [44].

Chapter 2

Touchscreen Visual Servo Hopping: A Low-Attention UAV Interface

2.1 Introduction

Twenty years ago, drones were complicated systems that mostly had been seen in the non-commercial world and applications like military operations and research [43]. Current progressions and innovations in equipment and software technologies enable the development of cheaper, smaller, more robust, and easier-to-control drones. This changes the application and use-case of drones, and now they are performing in a vast range of activities such as delivery (Figure 2.1), construction, surveillance, agriculture, and sports photography.

2.1.1 Roles of human in Human Drone Interaction (HDI)

There are different types of human-drone interaction, and hence there are different roles for humans in these interactions. The human role is dependent on the application and level of autonomy of the drone. Active control is the first common way of interaction [9] in which a human can send a command directly to the drones through an interface. Some examples of this kind of interaction are landscape photography with commercial drones and drone racing competitions. Having navigational sensors like lidar on the drone is not necessary in this case, as the user is flying the drone in his line of sight or remotely controlling the drone using the transmitted video from the drone.

The second role for humans in human-drone interaction is being a recipient. Human as a receiver usually is not controlling the drone. Examples could be receiving a box from a delivery drone [22] or watching an advertisement playing on a digital screen carried by a drone [9].

A companion relationship is another way of interaction. In this case, the drone might or might not be controlled by the human. A drone flying along with a human who is skiing on mountains or cycling on trails are examples of companionship [11].

Figure 2.1: Sparrow, a GPS-based autonomous navigated drone introduce by Drone Delivery Canada [3]

The last role for humans is a supervisor. Although drones can perform tasks autonomously with advanced autonomous navigation systems, there is still a need to keep the human in the loop in most applications. The human involvement, in this case, is either emergency piloting (i.e., controlling the drone in case of emergency) or scenario planning (i.e., pre-programming the behaviors).

In this work, we present a technique that facilitates drones' supervision in autonomous and semi-autonomous applications. We try to minimize the effort and attention needed from the supervisor by just requiring him to click on the next target of interest in the image-view of the drone's camera.

2.2 Related Work

Drones were initially used for military purposes and required highly qualified pilots to operate because of their complex user interfaces. As drone technologies become accessible, researchers begin changing the interfaces to no longer restrict the drones' control only via traditional remote controllers or ground control stations. In the following, we review different technologies used as an interface for human drone interaction.

2.2.1 Remote control transmitter (RCT)

RCTs are common devices for controlling drones. Figure 2.2 shows a commercial RCT for UAV and drone operations. RCTs have two sticks, and each stick can control two degrees of freedom (DOF) of a drone. Operators are needed to use both hands to handle and move the sticks. Usually, the left stick is used for upward and downward and rotations along the yaw axis, while the right stick is used for forwarding and backward and lateral movements. Controlling a drone using the RTCs is not intuitive, and operators need training before becoming masters of remote operation.

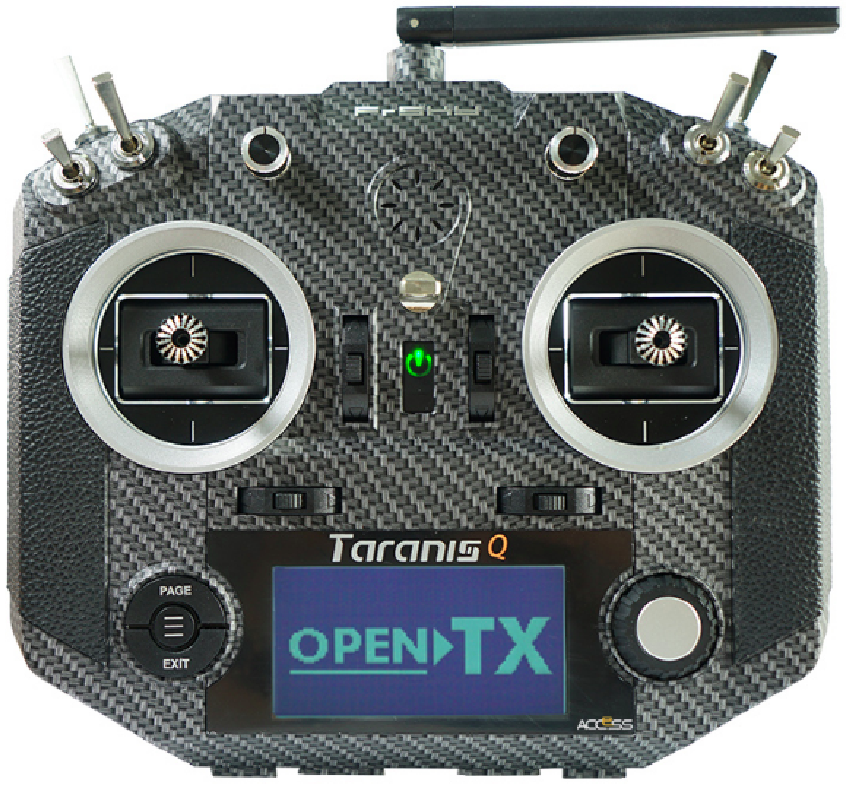

Figure 2.2: A commercial remote control transmitter produced by Frsky [8]

2.2.2 One-Hand Controllers

Controlling a drone with two hands can be very challenging for inexperienced users or people with physical disabilities as it requires the independent coordination of both hands. Another problem with convention RCTs is that the pilot needs to map the 2-D control of the joystick to the 3-D drone movements, which is not intuitive. One-hand controllers are novel joysticks built to tackle some of the discussed issues with the RCTs.

A Korean company has presented SHIFT [16], a new concept of a drone-controlling system. With this product, pilots can control a drone with a joystick held in one hand and includes a ring that users wear on their thumb.

There is another type of joysticks that is more like a glove. Wepulsit [45], a European company has developed an intelligent glove (Figure 2.3) that is used to translate the gestures of the user's fingers to the operational command for drone. These types of joysticks are still not very popular comparing to RCTs and touchscreen interfaces.

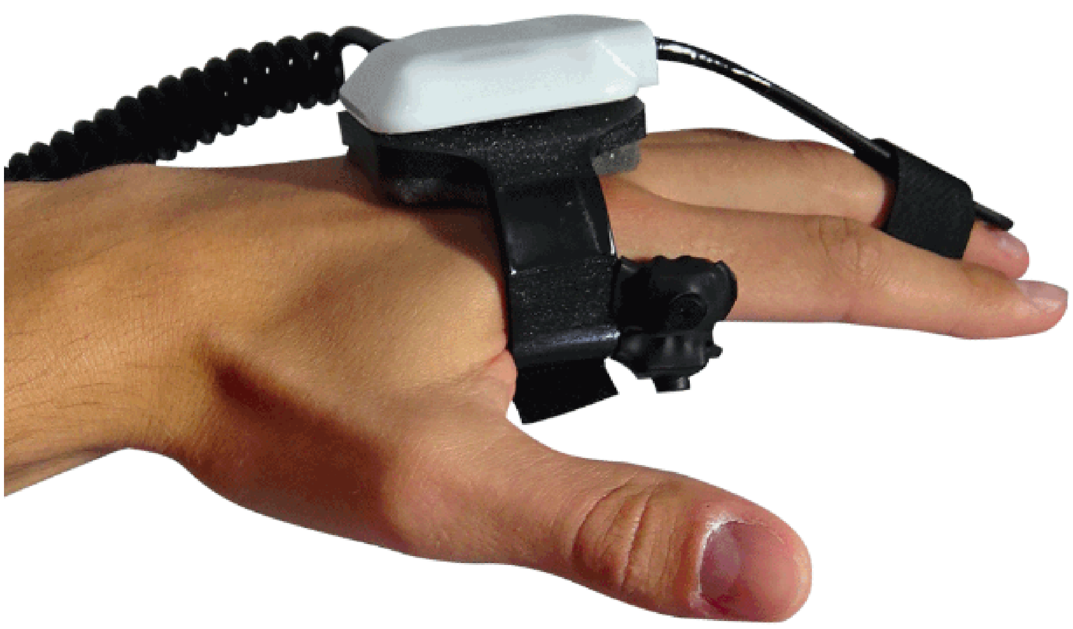

Figure 2.3: WEPULSIT one hand drone controlling system [45]

2.2.3 Tablets and Smartphones

With the advancement of WiFi technology, IoT, and smartphones, many emerging solutions use touch screen devices (i.e., smartphones and tablets) as a joystick for controlling robots. Touch-screen devices are used independently or in combination with remote control joysticks. In either case, an embedded screen enables pilots to watch the videos transmitted from a drone in real-time, and hence drone can be controlled even if it travels beyond the line of sight. In [17], a multi-touch screen is used to control the camera angle (using an attached motorized gimbal) and fly the robot in the environment.

Smartphones can be used without RCT by having the interface design consistent with RCT design. In this case, two virtual control pads replace the sticks of the physical RCT. The downside of using virtual pads is that there is no physical feedback to users' fingers.

In general, there is more flexibility in the interface design of the touch-screen devices than conventional RCTs, and hence they are being used extensively to control autonomous and semi-autonomous drones.

Skydio [15], and DJI [14] are two popular commercial drone companies providing their customers with advanced and robust drone solutions. Skydio introduces an enterprise controller joystick that consists of a built-in touchscreen, glove-compatible controls, and a new wireless system to extend the remote control range up to 6.2 kilometers. To track dynamic targets, such as humans on a motorbike, other than manual joysticks, Skydio introduces Skydio Beacon. Utilizing Skydio Beacon, the drone can track the object even when it can not visually follow it, and this works like a local GPS signal between user and robot.

DJI [14] also implements features for user interfaces to enable its customers to control drones conveniently. For example, Tapfly is a feature developed by DJI that allows users to send the drone in a specified direction, following a perfectly straight line, with only one click on the camera view while the drone's height is fixed. In this feature, the drone's obstacle sensing system constantly monitors the surrounding area and can navigate around anything in its path. If it comes to an obstacle that the drone cannot pass, it safely stops. Draw [13] is another feature implemented on DJI control interfaces for way-point control. It allows users to draw a route on a digital screen. Then, the drone moves in that direction with a steady altitude. DJI drones are trained to recognize various objects like people, bikes, cars, boats, and some animals. User can tap on the subject if he wants to track an object from the automatically recognizable objects. Otherwise, he is required to drag a box around it on the digital screen. The user also can set parameters like the height, distance, and speed of tracking. This feature is called Active Track in DJI's user interface.

2.2.4 Natural User Interfaces

Natural user interfaces (NUI) are other types of HDIs that do not use a joystick. A NUI enables users to communicate with drones through body gestures, voice, or brain-computer interfaces (BCIs) such as Electroencephalography (EEG). The goal of using NUIs is to work as close as possible to the user's expectations. It can potentially result in shorter training time, lower workload, and likely lower drone crashes [43].

In [27], multi-modal input signals, including voice command and simple body gestures like handshaking, are used to control the drones. Sepehr M. et al. [24] develop and train a convolutional neural network to detect the user's hand gestures and face up to 10 meters in the distance with high accuracy. Sepehr demonstrates that a drone can be controlled by simple hand gesture detection. In [4], a virtual interface projected on the ground by the drone can detect human gestures (commands), and the drone provides proper responses.

These interfaces impact the HDI in different ways, such as the required time for setup and training, precision, latency, and physical interaction distance. One limitation of controlling by a gesture using cameras installed on the drones is that it requires close physical interaction and maintaining the line of sight between drone and operator.

2.3 Proposed Method 1

As mentioned in the related work section, there are different types of interfaces for controlling a drone. Most commercial drone manufacturing companies like DJI offer remote controllers that either come with touch screens or are compatible with touch screen devices such as tablets and smartphones. This section proposes a new human-drone interaction method and explains how this method can help control drones in a semi-autonomous setup. The result of this work can be integrated with touch-screen joysticks.

Our goal is to minimize the time of active piloting, and for this purpose, we enable pilots to control drones by only selecting target objects in the camera image view consecutively to reach a destination. The objects may have different sizes or heights from the ground.

This work is inspired initially by previous research done in our lab by Monajjemi [26]. Monajjemi has demonstrated that a drone can successfully detect a human standing and signaling by waving arms. Once the human is detected, the drone tracks it and autonomously flys towards him.

The research [26] can be categorized under Natural User Interfaces because users can interact with drones by only showing a gesture, i.e., waving arms up and down. Although this work could be useful for some use cases such as rescue applications, it has limitations that block its usage by some other applications such as surveillance and monitoring. The main reason is that it requires the operator to be present in the camera field of view and close to the drone.

By considering the limitation mentioned above, we have repurposed the application to control the drone from a remote distance by selecting the target objects in the drone's camera image view. Our interface enables the operator to choose multiple targets step by step to reach a target in a very distant location that either is not originally in the image view initially or due to lack of visual features it is difficult to be detected and tracked. In the following, we explain the components involved in our setup in more detail.

The proposed system consists of three main hardware components (a drone equipped with a monocular camera, an off-board computer, and communication module) and three major software blocks (Behaviour state machine, Object tracker, and Controller). These components are explained in more detail hereunder.

2.3.1 Hardware Components

The Parrot Bebop 2, a lightweight consumer UAV platform with a monocular 30 frame per second camera with 640x368 pixel image resolution, is used in our approach. This decision's main reason is our team's previous experience, and its popularity in academic studies due to the open-source ROS packages [25] developed to supports communication, control, and monitoring. Easy-to-use software APIs facilitate quick prototyping and experiments.

The off-board computer (Core i5, 4G Ram) is used to address the required computation instead of the on-board computer on this robot because of not being adequate for our heavy computation. In this method, we have used a wireless module to connect the drone to the interface. This connectivity provides communication beyond-line-of-sight (BLOS) and enables the operator to control the drone remotely. Images from the camera and telemetry data are transmitted through this communication channel to the off-board computer and the user interface to select the targets of interest. The rate of this transmission is around 5Hz.

2.3.2 Software Components

We customized the three main software modules for our application. A behavior state machine is used on the top layer, consisting of idle, new target selection, tracking, moving towards the target, and lost states. The state machine is responsible for the drone's action and manages the interaction between the pilot and the robot.

We use the long-term visual tracker developed by [12] to track the operator's selected targets. This tracker applies a correlation-based algorithm for short-term tracking and tracking-learning-detection (TLD) framework for long-term tracking and loss detection.

We apply the same controller to navigate the drone towards the target as [26]. This controller is a cascade model that gets the tracked object's current location in the image plane as input and returns the roll, pitch, yaw, and altitude velocity as the outputs. This controller has different layers, including depth estimation, visual servo controller, and velocity controller layers. To estimate the depth (the distance from the drone to the center of the object), camera's intrinsic parameters, preliminary information of the object distance from the ground plane, and the camera's tilt with respect to the inertial frame of the UAV is used.

The visual servo is designed based on the classical IBVS controller [5] to calculate the drone's required velocity to smoothly approaching the target. The low-level velocity controller calculates the system's desired control values (pitch and roll) using the PI controller.

2.3.3 Experiment

To examine our method in the real environment, we have prepared a setup in which camera tilt angle and controller parameters are set based on previous work by Mani [26]. Our objective from this experiment is to demonstrate the new way of controlling the drone with our proposed interface, which does not require consistent operator's attention and reduces the operation time and effort. Secondly, we want to show our method's capability in approaching the invisible or hard-to-detect objects at a far distance by selecting and approaching multiple objects in a row (multiple hops).

Our target of interest is a 30*30cm dog 150m away from the drone, and our goal is to take a close-up image from this hardly visible object (hidden behind a sign). The user controls the drone by the interface to approach several other objects of interest in the middle. As Figure 2.4 shows, the drone is sent to the first object, a sign, then to a group of people, and finally to the dog once the dog is detectable and track-able in the image view. This experiment concludes that complex trajectories can be traced with very little user interaction by this proposed interface. The result of this experiment is presented in a poster at IROS 2017 conference. Figure 2.5 shows the selected pictures from the drone's camera while the red bounding box represents the tracking target in the camera view. As described, the first selected target is a signboard, and while the drone was approaching the first target, a group of people on the left side of the image was selected as the next target. And the dog, as the final target, was selected by the user once a) it appears in the image, and b) the size of the dog in the image was large enough that the tracker could accept it as the new target.

In the above experiment, the desired distance between the drone and the target (i.e., the dog) is set at 2.5m, and the drone's camera tilt is zero. In our experiment, the user can watch the live video streaming by the drone on a touchscreen display while robustly approaching the objects.

2.4 Proposed Method 2 (An improved version of method 1)

In the previous section, we demonstrated our interaction method with a drone utilizing a new user interface based on Mani's work [26]. Although there are advantages in using this method, there are serious limitations, and important assumptions in the base method [26]. We elaborate on these limitations below and provide a solution for each of them to improve the overall performance, enable more use cases, and have fewer method failures.

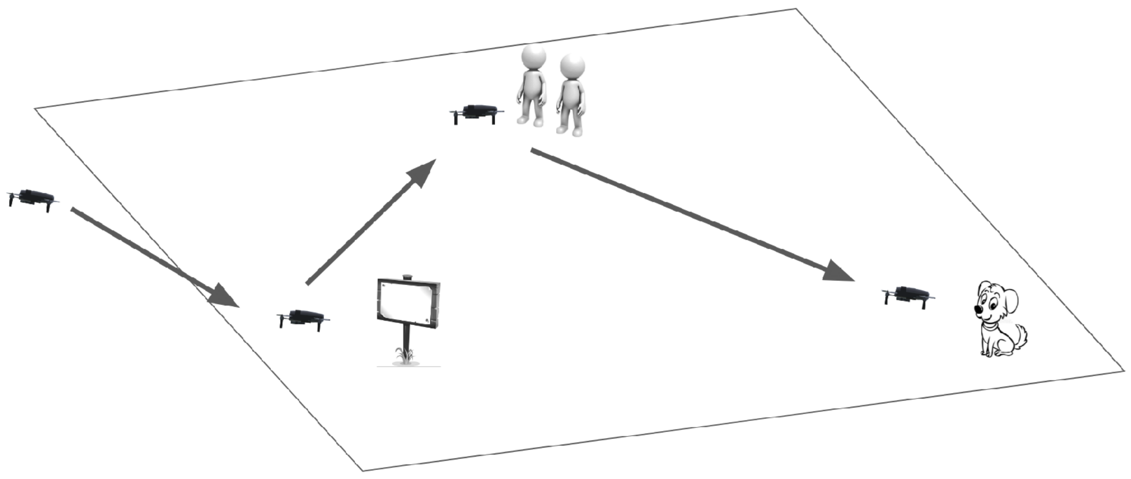
Figure 2.4: Schematic of the experiment: showing the drone's hopping to reach the last target. The actual images and targets of interest are shown in figure 2.5.

.

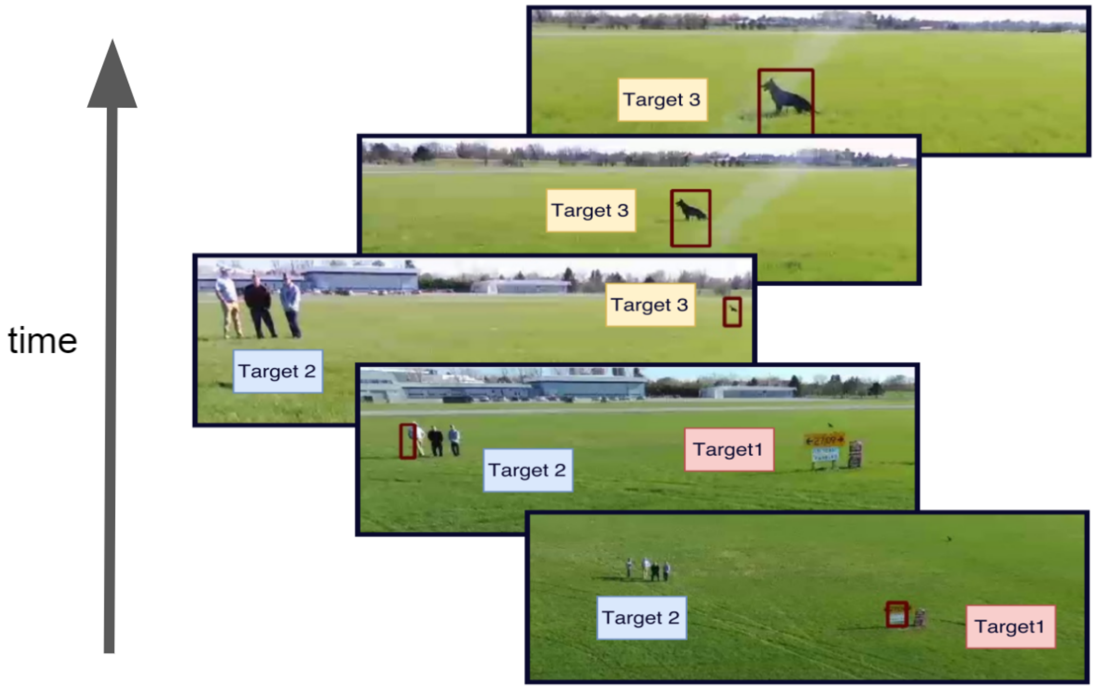

Figure 2.5: Real environment test: taking a close-up image of the 30*30cm object by flying 150m and only three clicks on the user interface [38].

.

The ground is assumed to be flat without a hillside or slope in the previous method, and the targets stand on the ground straight. To track each target, the controller relies on a number that the operator enters as the rough estimation of the target's height from the ground. Since the drone knows its distance from the ground by the IR sensor, given this assumption, the drone's relative distance from the object can be obtained. This assumption causes the method not to be applicable for the targets that are not standing on the ground. To overcome this assumption, we change the method to not depend on the height from the ground for the objects. We relax this assumption to just having prior on the height of the target. We can expand the usage of the interface to visual servoing to the targets in different heights like windows of a building. In this example (i.e, apartment windows), it is easier to estimate the target's height than estimating the height from the ground.

In this method, we use the prior knowledge of the target's height to estimate the robot's distance from the target. We then use the estimated depth to control the drone towards the target. In the next section, we describe how to calculate the depth given the prior on the height.

2.4.1 Depth estimation

To calculate the tracked target location using the monocular camera image, we use the prior knowledge of the target's height. Any point in the camera field of view can be projected into the image plane through camera intrinsic matrix K using the Equation 2.1 where u and v are equivalent to x and y position of a pixel in a projected image, and $[X, Y, Z]$ is the position vector of the point in the camera frame.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.1}$$

By rewriting Equation 2.1 using the inverse of the intrinsic matrix, one of the viable solutions for the position of a point in the camera frame can be calculated (Equation 2.2). $[X, Y, Z]$ is just one of the acceptable solutions for the point's position in the camera frame. In general, any point with coordination $[\gamma X, \gamma Y, \gamma Z]$ where γ is any positive number is a viable answer for the Equation 2.2. This a very well-known problem with monocular cameras, and it means for all the points located on a line passing through the origin of the projection (camera frame), there is only one projected pixel in the image. In the following, we describe how we calculate the target's actual position in the camera frame.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2.2}$$

Figure 2.6 demonstrates that we can calculate the actual position of a point in 3D if we know the actual depth of a point in the camera frame (Δ). By solving the Equation 2.3

given the actual depth of a point, Δ, the correct multiplier γ can be calculated. Hence, we can compute the location of the actual point $[\gamma X, \gamma Y, \gamma Z]$, in the camera frame.

$$\Delta = Z_{actual} = \gamma Z \tag{2.3}$$

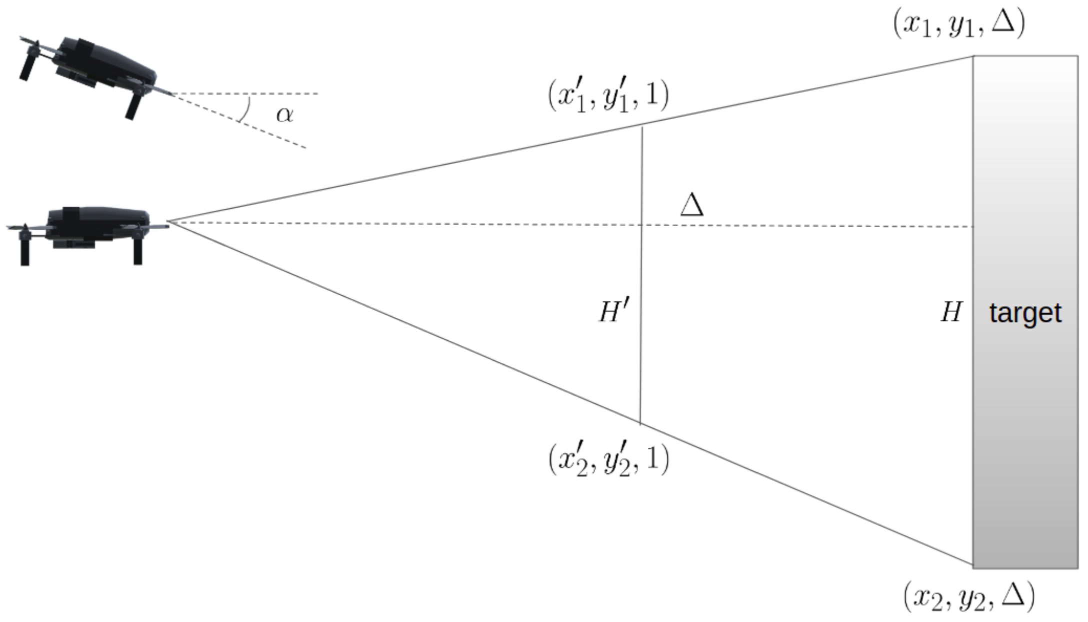

Figure 2.6: Target with height H and distance Δ from the robot. we can assume the target has 1m distance from the robot and calculate the corresponding height H'. This allows us to find actual distance to the target (Δ)

In Figure 2.6, $Point_1 = (x_1, y_1, \Delta)$ and $Point_2 = (x_2, y_2, \Delta)$ are respectively the position of the top and the bottom center of the target object in the drone's camera frame and are used in our visual servoing controller. We assume the target is standing perpendicular to drone's hovering plane and drone's pitch angle (α) is zero. Later, we will show that we can generalize the formulation to consider any non-zero pitch angle.

To calculate the position of $Point_1$ and $Point_2$ in the camera frame, we need their projected pixel position in the image, $[u_1, v_1]$ and $[u_2, v_2]$ respectively. As discussed above, using the inverse of the intrinsic matrix, a series of acceptable positions in the camera frame can be calculated for each projected pixel. At first, let us assume that the target's depth in the camera frame is $1m$. This assumption results in a possible coordination for the target's top and bottom center in the camera frame presented in Equation 2.4 and 2.5.

$$Point_1{}' = (x'_1, y'_1, 1) \tag{2.4}$$

$$Point_2{}' = (x'_2, y'_2, 1) \tag{2.5}$$

In the next step, by using Euclidean distance, the target's height is calculated based on the assumption that the target is in $1m$ distance from the camera (Equation 2.6).

$$H' = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2} \tag{2.6}$$

H' is only one possible candidate for the target's height. As it is clear in Figure 2.6, having the actual height H enables us to find out the actual target's depth Δ through Equation 2.7.

$$\Delta = \frac{H}{H'} \tag{2.7}$$

We can easily conclude that if the drone's camera were tilted by α, Δ can be calculated by Equation 2.8. Hence, we can generalize our solution to the cases where the drone's camera frame is tilted with respect to the hovering plane.

$$\Delta = \frac{H \cos \alpha}{H'} \tag{2.8}$$

The proposed depth calculation method might have some errors due to the human error in a) estimating the target's size and b) selecting the target's ROI accurately. Lets assume there are two rectangles associated with each target object (Figure 2.7). In this figure, The target has been shown in the center with a height of H. The first rectangle with a height of H_s demonstrates the target's user-selected ROI, which our tracking algorithm uses to track the target. There is a second rectangle shown in the picture with a height of H_e. This represents the user's estimation of the target's height when he is asked to insert an approximation for the height of the target. In Equation 2.7, we showed that the depth of the target is the ratio of $\frac{H}{H'}$. By considering the estimated and selected rectangles, we can use H_s to calculate H' and H_e to calculate H.

We showed that Δ is proportional to H. It can be concluded that the error in estimating H_e reflects linearly on the error in calculating Δ, which leads to the inaccuracy of the drone's final landing position close to the target. Figure 2.8 shows if H_e is a correct estimation for the height of the selected rectangular with H_s, drone travels path (b) and stops by the desired distance Δ from the target. Otherwise, based on the estimated height, the drone might take path (a) when H_e is greater than H_s, or path (c) when H_e is less than H_s, and end up landing in different positions to the target.

2.4.2 Experiment

Because of the limited access to the lab facilities during the Covid-19 pandemic, for testing this method, we have implemented a simulation environment using Gazebo-ROS tools, and the simulated model of bebop drone from [39] is used in our simulation. This model comes with a cascade controller that enables position control of the drone. The low-level controller

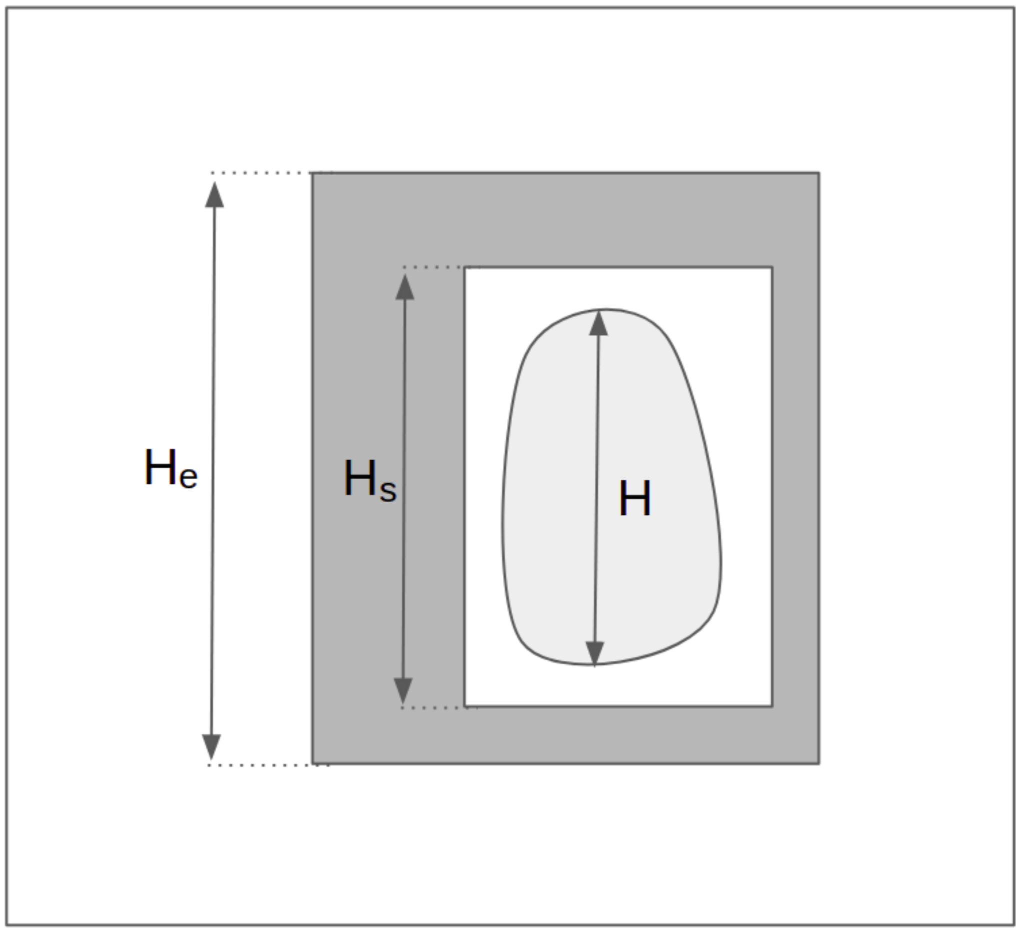

Figure 2.7: Object with height H, user estimation of height H_e, and selected bounding box with height Hs

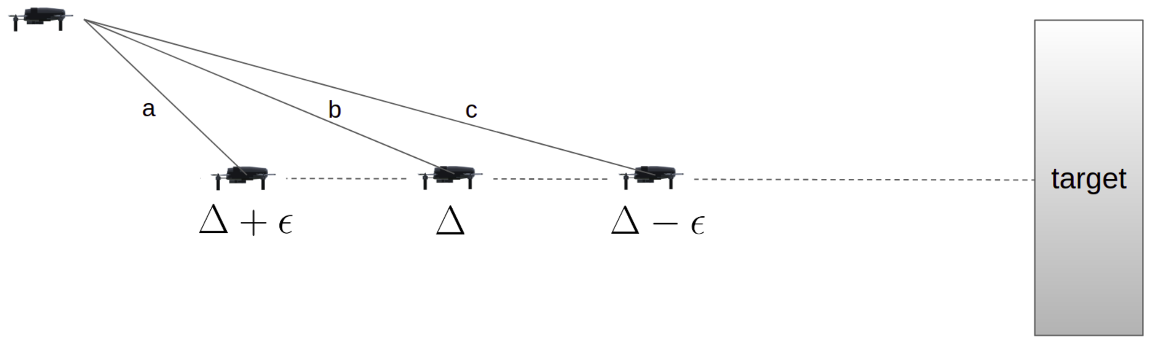

Figure 2.8: Error in drone stopping point resulting from incorrect estimation of selected bounding box height

is responsible for controlling the propellers' velocity and keeping the drone stable, while the high-level controller is responsible for taking the drone to the desired location. We also used the gazebo world model from [47] as our base world model but modified it to work with our testing scenario and simulated bebop-drone model.

In our approach, for the drone to reach the target, the user is required to define the drone's desired final distance from the target. As mentioned before, for some cases, specifically when the target's height is relatively large, the user can not pick a distance that is relatively close to the target because the target does not fit in the camera view once the drone reaches the point. To avoid this, we have implemented a constraint that causes the drone to stop if it reaches the user's desired distance or when the target reaches 3/4 of the camera image size.

In the following subsections, we explain different scenarios in which we have tested our approach. The results of each experiment are demonstrated and elaborated in more detail in each part.

Experiment 1: Track-ability

Before sending the drone towards the object, we need to identify inherent limitations with our approach. In this section, we set up several experiments that help us identify the geometric locations around a planar target in which our drone can track the target, fly towards it, and stop at the desired distance from it. To this purpose and also to better understand the behavior of the tracker used in our approach we run the drone manually on edges of a rectangle in x (depth) and y (lateral) plane in the simulated world with the fixed value in z (the center of the target height). Figure 2.9 shows an example in which our rectangle and the target in a 2D plane are drawn. Assume the target here is a 2x2 window of a building.

In this experiment, we set the window's center to be at position $(x = 0, y = 0, z = 4m)$ and the drone starts at point A with position $(x = 21m, y = -15m, z = 4m)$ and facing toward the building. Then, we fly the drone over the closed path [A, B, C, D, A]. The drone is kept at height $4m$ during this experiment since our focus for this part is to calculate the error resulted from moving in x and y direction. The rectangle is not symmetric to the target position in the y direction. This is due to the measuring error of target pose estimating in different directions of x and y.

During the drone's movement, the target object is localized in the camera frame. Given the drone's global pose in the simulated world, the target pose is estimated, and the error of this estimation is calculated. Figure 2.10 shows the error in estimating the z coordinate of the target pose (the target's center) over time. Although we tried to keep the drone at a fixed height due to camera turbulence, the z coordinate estimation is erroneous. The highlighted section in this chart shows the periods that our drone has stopped at each corner point. This result is interesting as we can see that the error is considerably small (<10 cm) during the time the drone is flying along the x direction. We conclude that the reason is

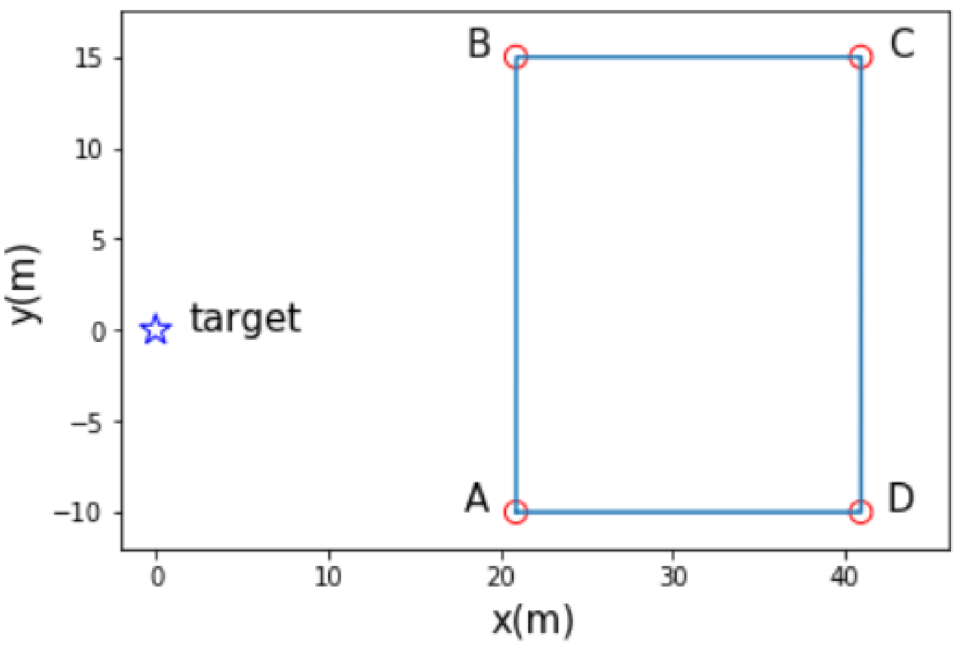

Figure 2.9: The drone is manually controlled over the path [A, B, C, D, A] tracking the target which is placed at $[0, 0]$ in a 2D plane.

the simulated model of this drone that we have used in our simulation. It shakes when it hovers or moves toward the side from point A to B and the reverse path C to D. However; it moves very smoothly when it flies forward (path D to A) and backward (path B to C).

In the same experiment, the error of estimating the x and y coordinates over the time [Figures 2.11 and 2.12. Figure 2.11 shows that by moving from point B to point C, the distance between the drone and the target is increased, so did the error of estimating the x coordinate of the target. On the other hand, the error is decreased by approaching the target (path D to A). Lateral flight in both directions increases the error of estimating x coordinate.

In Figure 2.12, lateral movements from point A to B and from point C to D increase the value of the error of estimating the x coordinate of the target. As shown in this figure, the big jump occurs when the drone is far from the target and moves laterally from C to D. The source of this error is the camera turbulence and image distortion, which affects more on the error in greater distances. The drone's dynamic enforces the vehicle to roll while moving along the y direction, which causes the distortion.

Considering the results of this experiment, we conclude that the drone's heading has to be kept facing toward the target all the time to minimize the estimation error resulted from the lateral movements. We also have seen that the drone's shaking can increase the error in estimating the target's z coordinate. Given this conclusion, we have performed more tests on hovering the drone to capture the effect of shaking on the tracker. We have monitored

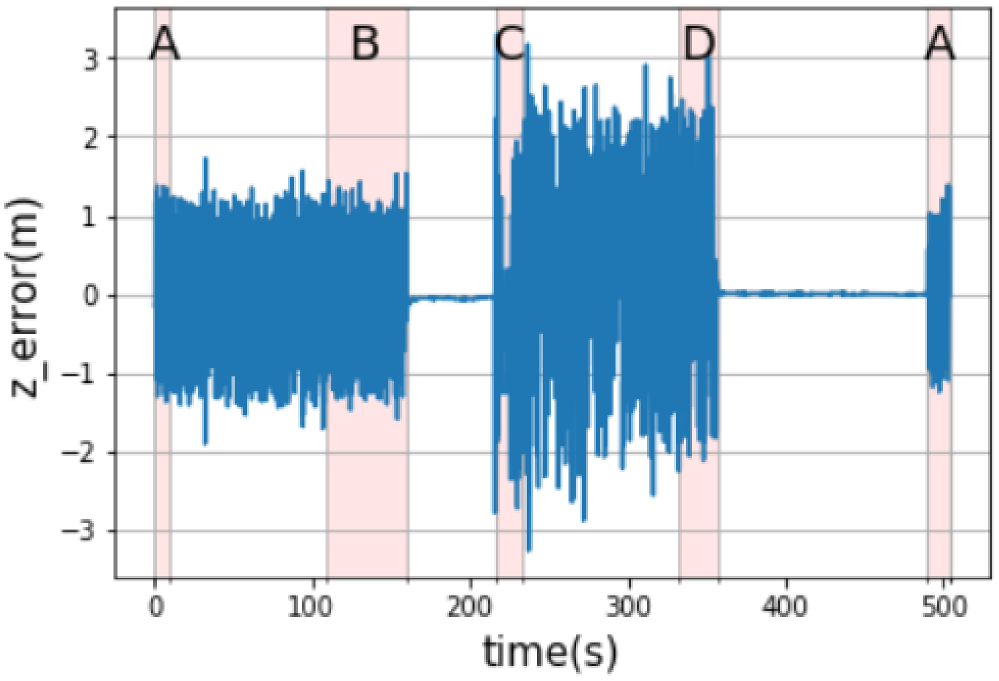

Figure 2.10: The estimated error of the target's z coordinate during the flight time over the rectangle edges.

the camera image, and we concluded that this drone's camera turbulence could shift the image view by 100 pixels in the z direction. This imposes a constraint on the height of the tracked targets in the image view and, therefore, a limitation on the object's maximum size that we can track in the real world by our approach.

To be safe, for tracking, we can focus on selecting the targets which have at least 100 pixels gap from the top and the bottom of the image view. The camera image's resolution in our simulated drone is 1080 pixels on the z axis that means the maximum height of the object in the camera image has to be $(1080 - 2 * 100)/1080 = 0.81$ of the image.

On the other side, our tracker fails to track tiny targets due to the lack of having rich features in the image. Quantifying the minimum size of the object is complicated and depends on the object's features and the environment surrounding it. There are two reasons that the target looks very small in the image view of the camera. One obvious reason is a significant distance between the drone and the target, and the other one is observing the object (especially planar objects such as a window or QR code in our experiments) from a very sharp angle in the camera results in a projection with smaller width in the image. We have done some experiments by selecting multiple objects in the simulation environment and testing different bounding boxes. The results helped us estimate the minimum required size of a bounding box around an object in the image so the tracker does not lose it. We conclude that the bounding box has to be at least 75x75 pixels to have robust tracking.

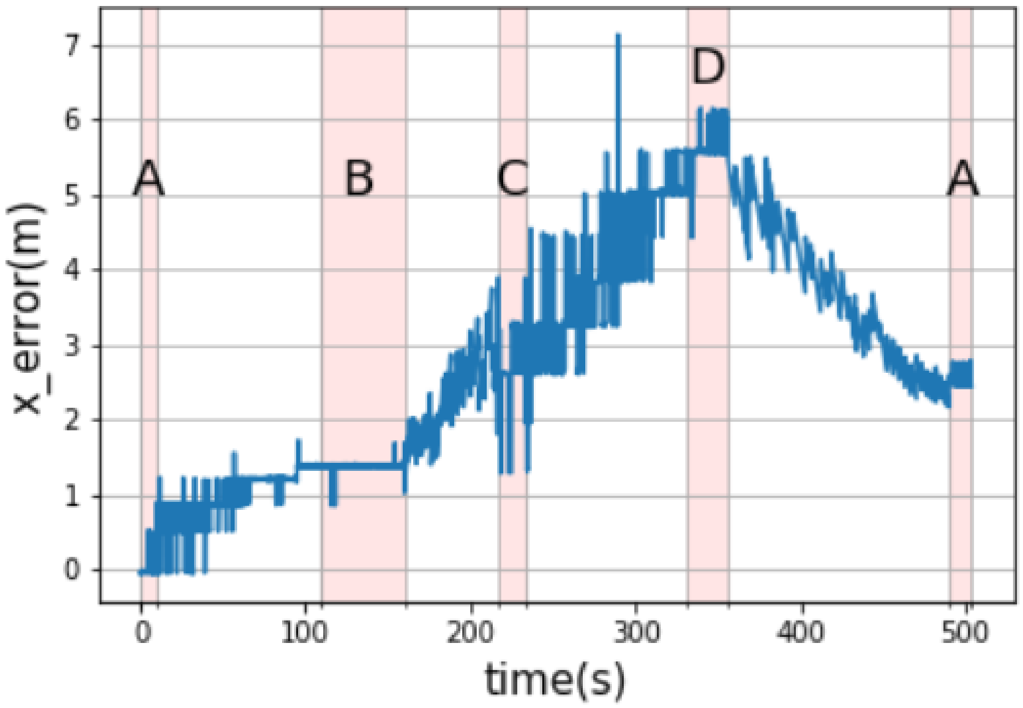

Figure 2.11: The error of estimating the x coordinate of the target during the flight time over the rectangle edges.

Considering the limitations on the object size in the image frame, we can compute the track-ability geometric locus around any target. For example, there is a planar object in our simulation world with the size of 1m x 1m installed on a wall with its center located at location $(x, y) = (0, 0)$ at any height from the ground. We aim to calculate the area (locus) around this point from which if our drone starts while facing toward the center of the object, the tracker can track the object robustly. Figure 2.13 shows this area for the above-mentioned target. This area looks like an oval with missing a piece that shows the drone can not get closer to the target because of our constraints imposing a maximum threshold on the image's ratio that the object occupies. The area outside of the oval shown in pink is very far from the target affecting the size of the bounding and, therefore, on the object's track-ability. Also, the blue area shows the locations that are either too close to the target or are at a very sharp angle to the object, causing smaller projection in the camera image.

Experiment 2: Multi-Step Tracking

This experiment demonstrates an example of a 2 step tracking of an object which is very far from the drone's initial position and can not be achieved in one step. Figure 2.14 shows the setup environment for this experiment in our simulation world. In this example, the user is

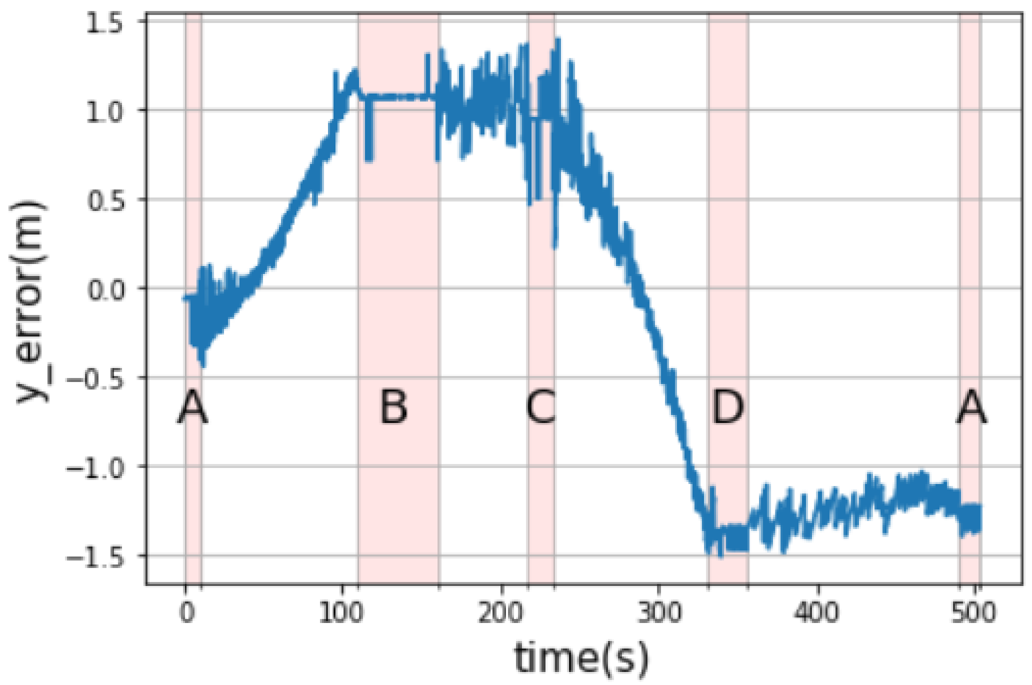

Figure 2.12: The error of estimating the y coordinate of the target during the flight time over the rectangle edges.

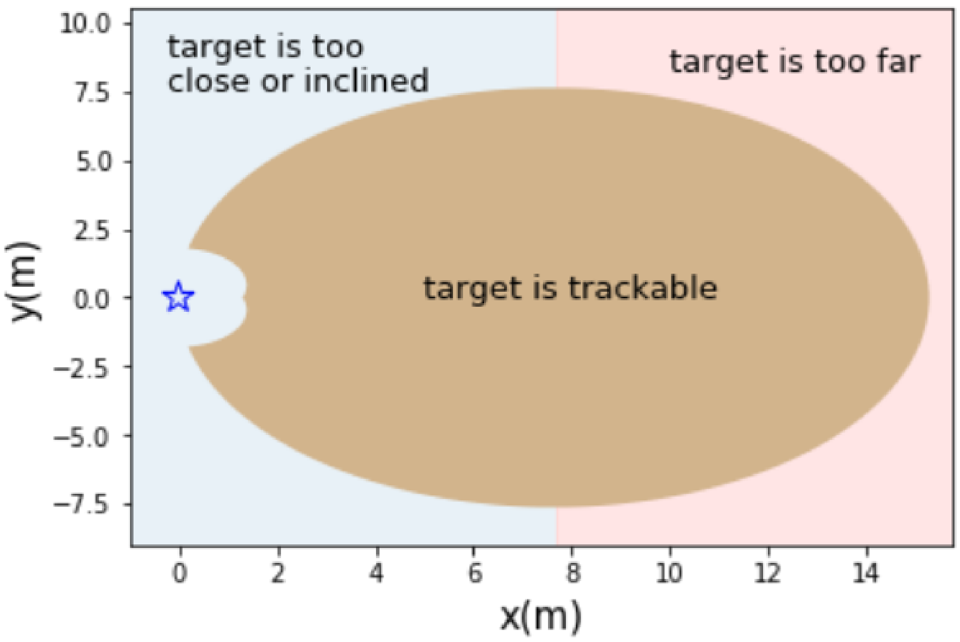

Figure 2.13: Track-ability area for a planner object with 1m*1m dimension and located at $[x = 0, y = 0]$ in a 2D plane

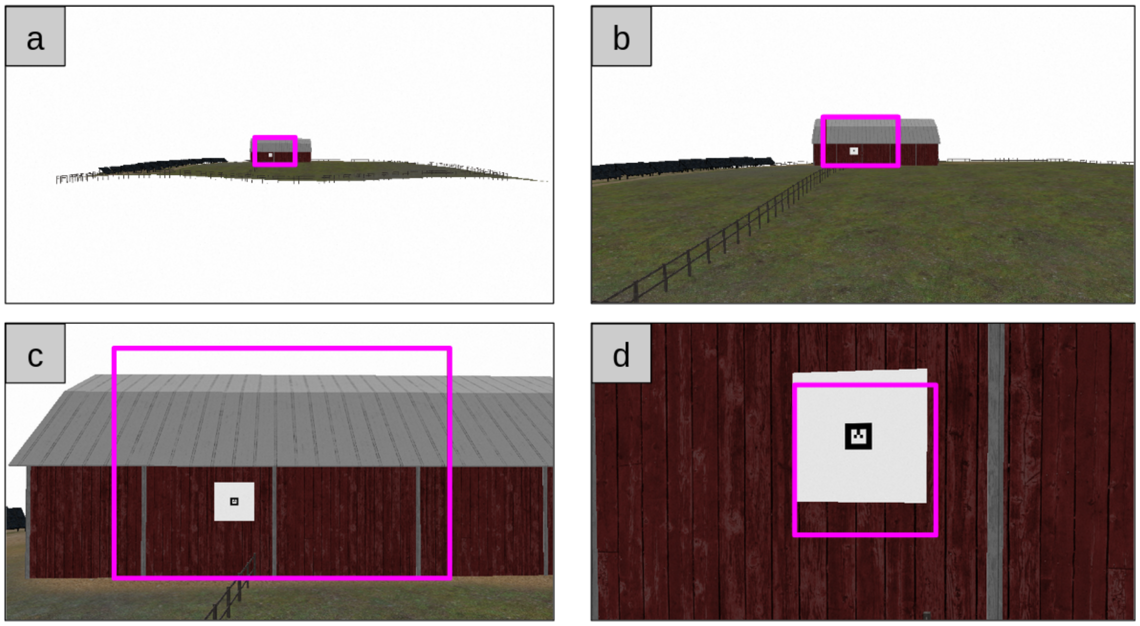

Figure 2.14: Images from drones camera while it has been sent by a user to take a close up picture of an AR tag. (a) building is selected as a first target from 190m away. (b) drone tracks building and flies towards it while it controls its height. (c) drone reaches into the building track-ability limit and user picks the window as the next target. (d) drone reaches to the desired distance (3m) from the window, stops and take a close-up photo of the AR tag

required to first select a bigger object for tracking (i.e., the building in sub-figure (a)) and then, when it is closer, to select the main object (i.e., the window installed on the building in sub-figure(c)).

We have used the ar-track-alvar package [46] in our simulation to generate an AR tag and used our interface to send the drone toward it to take a close-up photo of the tag. The AR tag's height, the window, and the building are 0.4, 2m, and 13m, respectively.

The drone is sent to this mission by the minimum required attention from the pilot. From the discoveries of the first experiment, in Figure 2.15, the calculated theoretical track-able area for both targets (i.e., the locus around the window in green color and the building in pink) is shown. As the figure illustrates, there is a limitation on how much the drone can get close to the targets. The source of this limitation, as described in the previous experiment, is the required portion of the camera image occupied by each object of interest, and also we need to make sure we save enough room for the drone's shaking (i.e., at least 100 pixels from the top and the bottom in the camera image).

Since our approach enables tracking of the objects with various height and distance from the ground, we have adjusted the drone's controller parameters so that the height (z coordinate) of the drone merges faster to the z coordinate of the target compared to its lateral (y) and depth (x) difference. The reason behind this decision is to try keeping the object in the center of the camera image because as we get closer to the target, the chance of losing the target in the image frame is increasing. Also, the drone's heading is toward the tracked object to decrease the error in the target pose estimation resulted from the drone's shaking. This helps to avoid lateral movements and selecting the shortest path to the target.

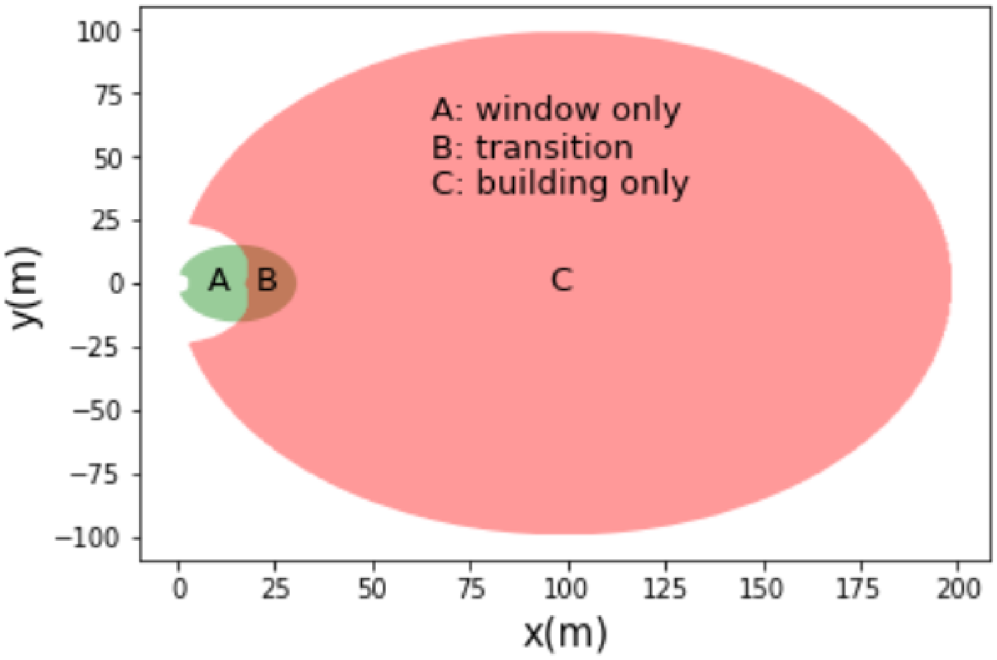

Figure 2.15: Track-ability area for the building and the window with 13m and 2m height respectively

We have selected different starting points (inside and outside of the theoretically calculated locus in Figure 2.15) in which the drone has sent toward the targets. We illustrated the initial positions (marked with a star) and the paths the drone takes (blue trajectories) in Figure 2.16. The positions that the drone stops after tracking the first object (i.e., the building) are marked with a small red circle. These are the points that the drone can not move forward as the bounding box ratio around the building selected by the user is greater than the threshold we set (0.82). At these points, the user selects another bounding box around the second object (i.e., the window). The acceptable distance from the target is set to 3m.

The initial points are selected randomly out of the track-ability area and inside it to test our method's robustness. As we can see in Figure 2.16, most of the cases with initial points outside of the pink area are failed due to the tracker's poor performance. In contrast, when the initial position is inside this area, the controller can control the heading toward the target and keep the drone in a straight line and reach the destination. All the trials with this constraint (5 out of 5) were successful. However, for cases with initial positions outside of the track-ability area, the trials failed in most cases (4 out of 6).

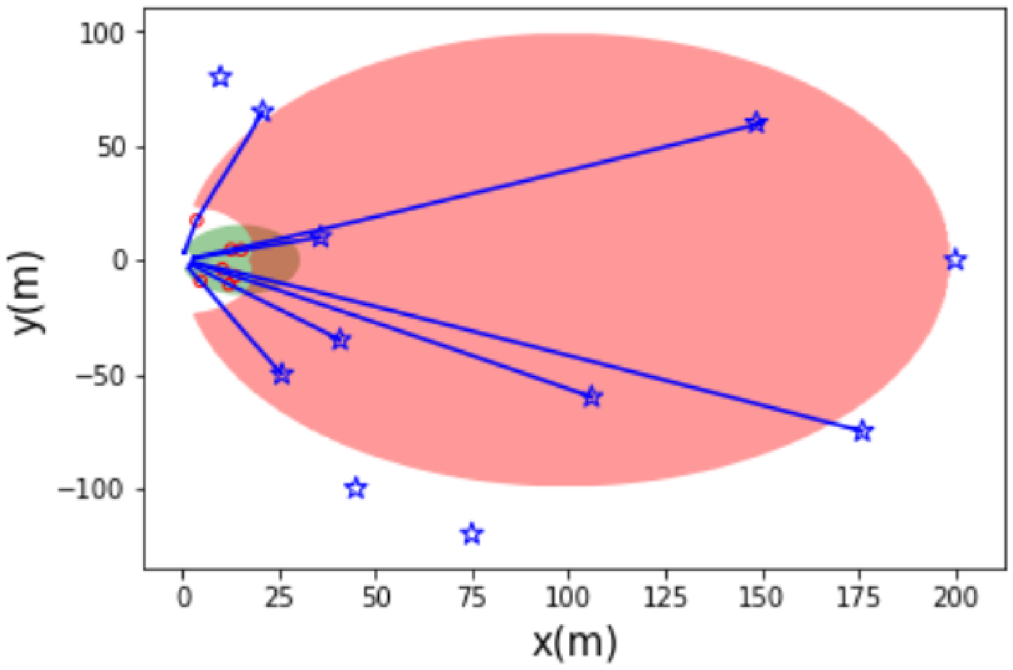

Figure 2.16: Result of experiment: Drones trajectory projected in 2D track-ability plane. Drone tracks building from star points to circle points. Then user selects the window at circle points as the next target

2.5 Conclusion, limitations and future work

In the first section of this chapter, we have introduced a new method (i.e. a new interface) that helps a pilot semi-autonomously control a drone toward a target from a far distance by selecting a bounding box around the objects of interest. We have tested our method in the real world to approach a dog with 30cm height starting from 150m away by only selecting a few targets in this path. The limitation of this approach is that the user is required to estimate the object's height from the ground; hence it is not easy to track the objects not standing on the ground.

In the second method, we have introduced an improved version of the interface using a new controlling algorithm to estimate a target's position by relying on a vision-based tracker. Using the proposed interface, a drone can approach targets located at an arbitrary height from the ground. In this method, we only require the object's height (i.e., the length along z axis) as the prior. Through different scenarios, we have experimented the method and have indicated that it can successfully predict the target pose based on the object's height. We have explained the results of our experiments employing a newly defined concept of track-ability. It helps us determine the geometric area around a target in which a drone can start approaching toward a selected target and be controlled successfully to end up stopping at an acceptable distance from the target. All the limitations, such as camera shaking and targets' size in the image view, are considered in our track-ability area.

Our second approach is tested in a simulation environment. We have shown that our simulated drone follows the track-ability concept that we introduced. In other words, when the drone is located inside the track-ability area of a target (i.e., the building), the drone can successfully track and approach the target. In this simulated experiment, our drone can take a close-up photo from the AR tag that is only 40cm in height by only selecting two targets and running a drone semi-autonomously from more than 190m away. This experiment also proves that targets with an arbitrary height from the ground can be tracked.

There are some existing limitations with our approach and some potential improvements for the future, which are explained hereunder.

2.5.1 Semantic objects

In our proposed method, the supervisor pilot is required to estimate and insert the height of each selected target. This step can be skipped using semantic object detection and the prior knowledge on the average height of objects. For example, the drone can automatically set the height to average human height whenever a human has been selected. However, this might not be practical for all objects. Buildings, as an example, can have different heights. The other use case for using semantics in our application is to adjust the selected bounding box. In some cases, when we want to draw a bounding box, it does not perfectly fit the target object. If it is a semantic object which is already known to the machine, we can have an algorithm to refine the bounding box.

2.5.2 Obstacle detection

We do not detect the obstacles in the proposed method. To enable the drone to avoid arbitrary obstacles in its path to the targets, either it can be equipped with range sensors such as IR, depth camera, and Lidar or semantic heavy computational algorithms to detect the obstacles in-camera image. The later option is possible thanks to recent progress in the development of deep networks. Today, robots can learn many things from observing a large amount of data, although network-based algorithms are usually susceptible to the domain of data they have been trained with. In that fashion, if these algorithms are used, it may need to retrain them with data captured from the test environment.

Chapter 3

Semantic Visual Teach and Repeat

In this chapter the use of semantic object detections as robust features for Visual Teach and Repeat (VTR) is presented. This work is motivated by the application of autonomous navigation in an GPS-denied environment where only visual information of a path is presented to the robot, and the robot should reliably repeat the same path. This work was the published 15th Conference on Computer and Robot Vision [44]. One of the main challenges in this task is to design a system in a way that is robust to normal environmental changes such as lighting, appearance changes and camera viewpoint changes. Visual Teach and Repeat (VTR) has become a canonical task in robotics, and has many practical applications including surveillance patrols, inspection of structures and transporting goods. VTR has been studied using land, air and water vehicles. Here we consider monocular VTR on Unmanned Air Vehicles (UAVs).

VTR has two phases. In the teaching phase, a map-like memory is recorded from observations while the robot (or other training device) is piloted over the desired path. Then in the repeat phase the robot must (re)localize itself on the prior path, and repeat the remainder of it based on the prior observations in memory.

To investigate this problem we can break each phase into parts. Teaching phase is basically to encode visual information and store them in memory. These information later will be used by localization and the motion planner. Repeating phase can be broken down into two parts: *relocalization* and *motion planning*. A robot cannot effectively repeat the thought path without having a reliable localization. Once it can relocalize itself on the path, it can plan the motion accordingly. A reliable motion planning method is vital to the system too. Since taking a wrong action can take the robot far out of the path and this may lead to localization failure and the robot to be lost.

As discussed above all parts of a VTR system is important to be carefully designed. Visual information encoder in the teaching phase and relocalization in repeating phase are paired with an extra requirement of the visual encoder to be informative for motion planner too. The two main categories for relocalization are (i) to make a 3D map using SLAM, with landmarks in 3D space; or (ii) to record a sequence of keyframes, with landmarks in image space. Approaches for the motion planner are vary and our choice on motion planner depends on the localization method and the type of information stored in memory.

Our approach to problem of encoding visual information into memory is to use only sparse semantic object features. Recent work on Convolutional Neural Networks showed significant success on training models that can detect multiple objects of tens or hundreds of categories in an image at frame rates. We show that such detections are repeatable enough to use as landmarks for VTR, without any low-level image features. Since object detections are highly invariant to lighting and surface appearance changes, our VTR can cope with global lighting changes and local movements of the landmark objects as well as the camera viewpoint changes.

With such a powerful and robust visual encoder we could have the same set of information in different lighting situation, different seasons and environments with dynamic objects within where most of low-level feature detectors fail to detect the same set of features. Regarding the selection of semantic object detector as our visual encoder, we used overlap of the objects" bounding boxes with the same class label as a measure to estimate the similarity of two keyframes. Spatial-temporal constraints inspired by Sequential SLAM [23] are also applied to enhance the accuracy and performance of relocalization. Using a temporal sequence of landmarks reduces the effect of perceptual aliasing caused by sparse landmarks and a limited landmark vocabulary.

Our choices of motion planner is limited because of the chosen visual encoder method. During the repeating phase motion planner will acquire current observations of the environment as well as the desire keyframe provided by relocalization. These observations are bounding boxes of detected objects with their class labels in both keyframes. By the two, motion planner should take an action that will pass the robot through the thought path. Even though the visual information are powerful enough to manage camera viewpoint changes and different environmental conditions, but taking a bad action may fail the system entirely. Our motion controller is based on the *funnel control* theory [6] with a novel extension that exploits the scene memory to anticipate the future.

Later in this chapter we will discuss work related to semantic visual teach and repeat in 2.1. Our method and system design are presented in 2.2 followed by experiments in 2.3. We will finally finish this chapter by a conclusion in 2.4 and discussion and possible future works in 2.5.

3.1 Related Work

Barfoot has a significant body of work on Teach and Repeat navigation, including [7] which tested a ground robot in a path over 1km, demonstrating a color-constant method to handle changes in illumination. However, this work is sensitive to changes of viewpoint and hence requires the initial position to be similar. To handle gradually-accumulating environmental changes, [21] employs a “multi-experience" method for VTR, which was introduced in [31].

Visual teach and repeat using UAVs is only recently feasible, with few early examples [32] [29]. In [32] the authors demonstrated a quad-rotor with downward-facing camera that repeats a simple path starting from the same position with no changes in the environment. The method is similar to [10] and based on SURF descriptors and pose-based localization. Nguyen et al in [29] also choose SURF features for descriptors but their localization technique is appearance-based. Experiments are in simulation, using an AR-Drone model in Gazebo, and again the environment is unchanged between teach and repeat.

Surber et al in [41] build a map using structure from motion (SfM) from data captured in the teaching phase and execute visual-inertial odometry on a real UAV in the repeating phase for localization, using a previously constructed map as prior knowledge. They use BRIEF descriptors and report that relocalization is successful only where the appearance is similar enough to the reference map.

Krajnik et al in [18] examines combinations of feature detectors and descriptors for VTR, favouring the STAR feature detector and GRIEF feature descriptor for good accuracy with computational speed. GRIEF is a binary feature descriptor intended to be robust to seasonal changes in appearance.

So-called “semantic" segmentation of foreground objects from images is a traditional topic in computer vision and robotics [35], [37]. The recent development in deep neural networks, particularly CNNs has this area flourishing [40, 28, 42, 2]. New object detection algorithms provide much better accuracy and speed than was previously available [33, 34, 19] which inspired our attempt to use them as online real-time feature detectors

3.2 Method

3.2.1 System Overview

Figure 3.1 outlines the structure of our system. During teaching, the UAV is flown or carried along the target trajectory. VTR system runs on a stationary computer equipped with GPU along with YOLO-2 CNN object detector. it receives video frames from the UAV periodically, passing them to the detector. The CNN output is processed to obtain a scene descriptor of the locations of objects and their class labels in the image. Each descriptor is stored in sequence. The finished sequence is the robot’s memory of the learned trajectory. The rate of storage set to be real-time to make sure all visual data are being stored in memory. However, for memory efficiency and processing time of the system, a non-real-time rate can be set. Trade-off between choosing a small and large frame rate is to have all important visual sensory information or have a smaller memory and faster relocalization.

In the repeat phase, the UAV motion is actively controlled by the VTR system. Again, images from the robot input to the object detector to obtain a scene descriptor. The re-

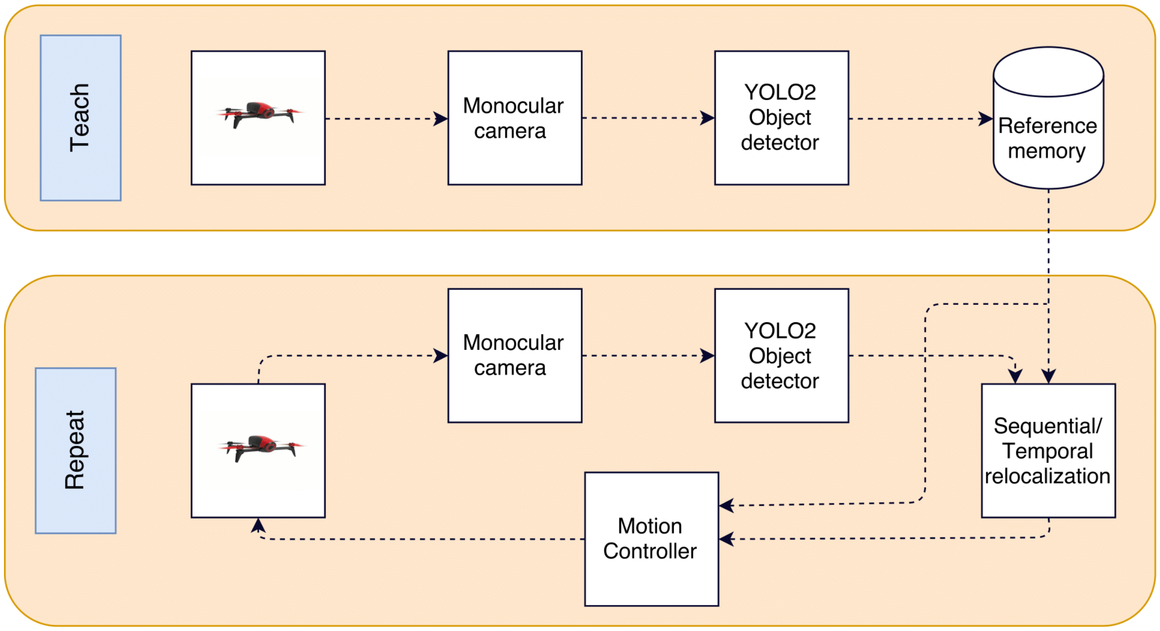

Figure 3.1: System Overview [44] (Copyright ©2018, IEEE)

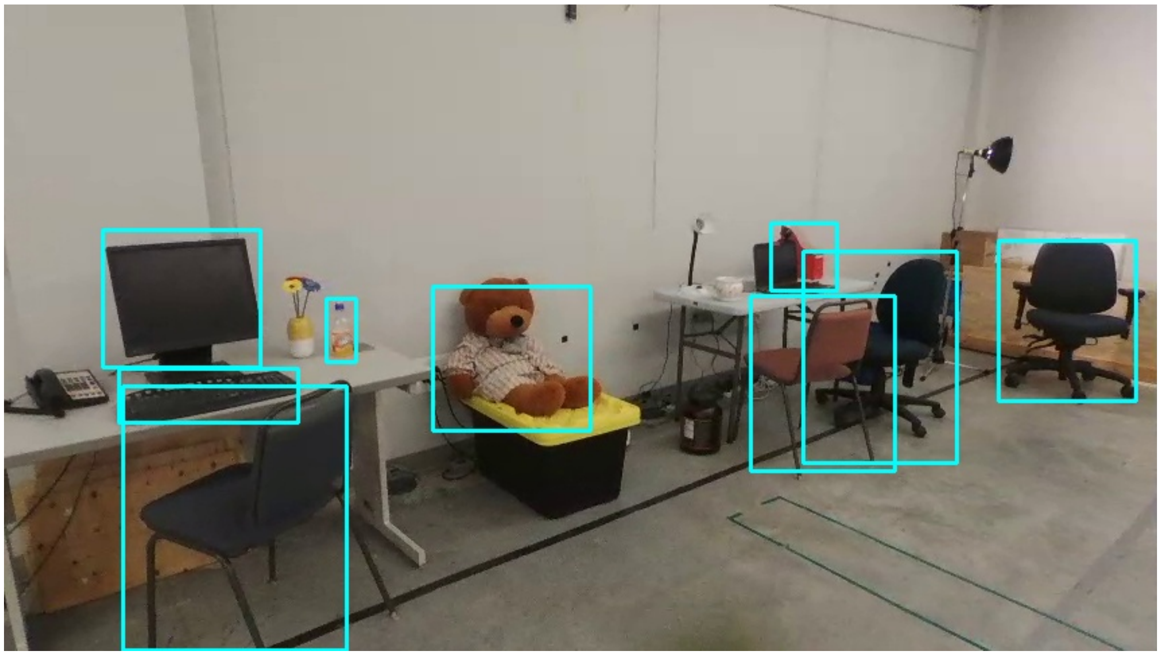

Figure 3.2: YOLO-2 predictions on an image with object labels predicted correctly.

localization module then finds the closest match between the current scene descriptor and memorized ones. Once localized, motion controller attempts to move the robot so that the next scene will look like the next scene in the stored sequence. This action will be sent back to the robot to be taken. Communication between the robot and stationary computer is over WiFi. These modules are described in more detail below.

3.2.2 Detection

The YOLO-2 [33] CNN object detector is being used for object detection. This model is trained on the COCO dataset. It is notable for its speed in comparison with other deep networks like Faster RCNN [34]. We can detect objects at 40fps on a commodity PC with NVIDA-GTX 1080Ti GPU.

YOLO-2 has different pre-trained models that are vary on the number of object categories, accuracy, size of models and processing time subsequently. It can detect a variety of common objects and animals, Figure 3.2 shows an example of detections on an image made by YOLO. YOLO was not designed for UAV landmark detection. We use YOLO and an office/lab environment that contains objects detected by YOLO as a placeholder for a dedicated UAV landmark detector and outdoor flights in this initial study. The system should work as described without modification in real-world application environments given an appropriately trained feature detector. For example, VTR across a city park might use signposts, path junctions, benches, fence posts, flower beds, tennis courts, and swimming pools as semantic features.

3.2.3 Reference Memory

The object detector model provides a bounding box, confidence and class label for each detected object. Detections below a confidence threshold are discard (we used confidence $\geq$ 0.55 in our experiments). Some object labels that are not reliable landmarks such as *person* and *cat* are discard too since their world positions are not fixed. Each detection is stored as a vector of five floating point values describing the object category and bounding box in the image. We found that each scene might contain two to eight objects, so the resulting whole-scene descriptor is small: of the order of $5 * 8 * 4 = 160$ bytes. For comparison single original SIFT point descriptor is 512 bytes (both using 32-bit floats), and in typical use scenes contain 500 or more points, for 256,000 bytes per scene. Our scene memory is thus around 1600 times smaller than a SIFT-based method while achieving reliable VTR.

3.2.4 Relocalization

So far we discussed how a reference memory will be created upon recording visual data during the teaching phase. Based on the data stored in reference memory, the robot should relocalize itself within the sequence of descriptors in reference memory. However, Robot's initial location may be far from the original taught starting location. Also, in a real world application, environment may change over time. For example some objects may get removed or moved around, lighting may be different during a day. In all of these cases, the robot should still reliably find the correct match in order to take a proper action.

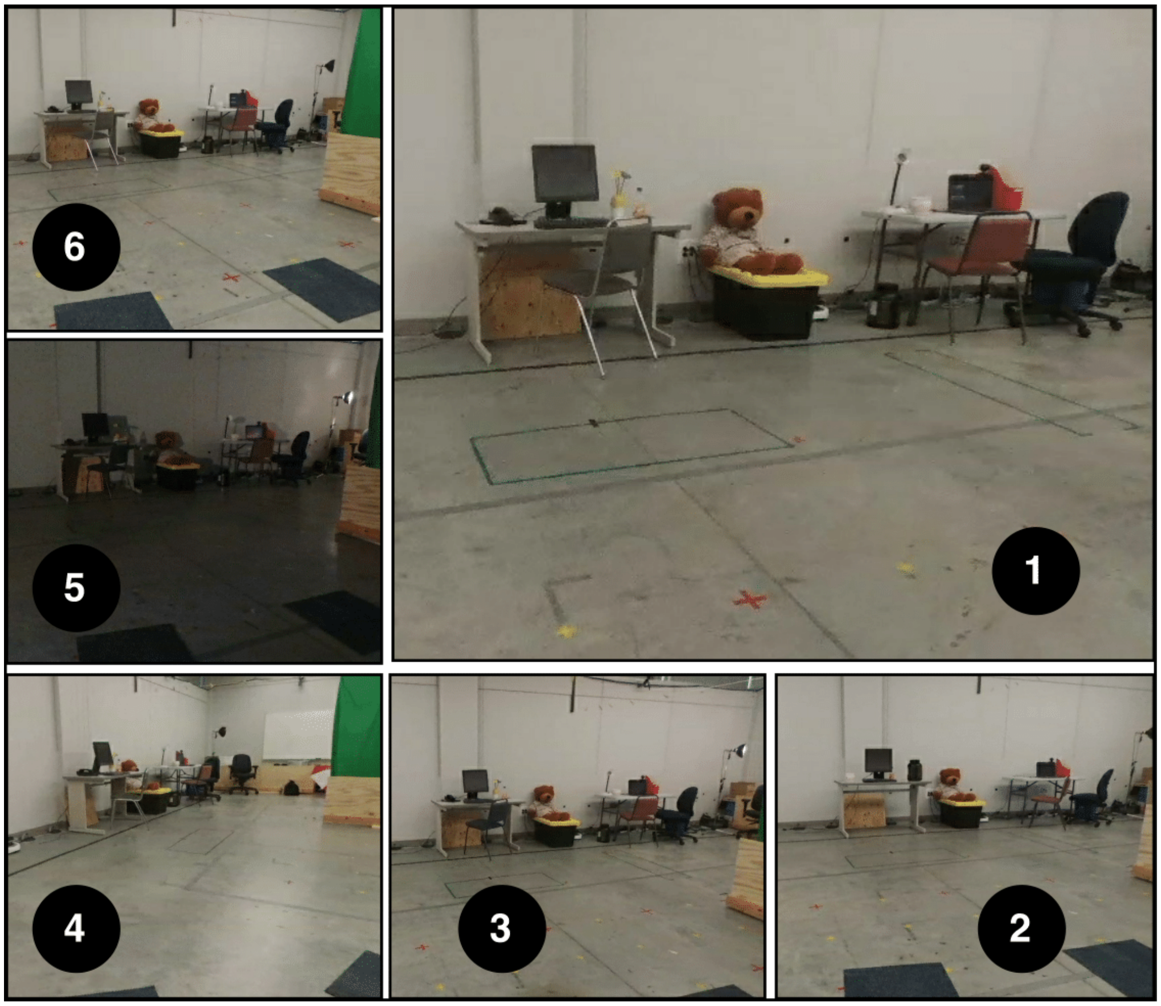

Figure 3.3: Relocalization robustness: (1) is the first image in the taught sequence; (2-6) are examples where the robot has successfully relocalized despite large changes in viewpoint and scene contents and appearance by matching a sequence of object locations (chair, screen, bear, etc.) from a sequence of descriptors stored during teaching and correctly located (1) as the closest matching image. Note the missing and moved objects in the repeats. [44] (Copyright ©2018, IEEE)

Similarity measurement used in our approach is first to find the similarity for two individual object detections by calculating area of overlap of their bounding boxes to the sum of their bounding box area:

$$S_O = \frac{(\alpha s' \cap \alpha s'')}{\alpha(s' + s'')} \tag{3.1}$$

Where s' and s" denotes the area of the bounding boxes of objects recorded in the target and current images respectively. α is a weighting factor to tune matching sensitivity. Two individuals may not have an overlap between their bounding boxes because of many factors including environment changes, taking wrong actions and leaving the path or starting from a different location. Having an α greater than 1 in equation 3.1 results in increasing the size of bounding boxes so that an overlap may occurred. We chose this by experiment, to get a working value of 4.

Object association is a big issue in the case of having a descriptor that has two or more objects with same class label. Distance of objects in two frames and their size is a good factor to find the association. Since the similarity measurement conveys these factors implicitly, object association in our approach relies on the similarity of two objects. We first calculate all the similarity of objects with same class label and then ones with higher similarity will be associated. Overall similarity of keyframes is then calculated by averaging over all the similarities (S_O):

$$S_I = \frac{\sum_{n=1}^{N} {S_O}^n}{N}, \tag{3.2}$$

where N is the number of objects in the scene.

Compared to point-feature approaches, we have very few features in each scene, and a dictionary of only a 80 object categories. Thus scene descriptors are not highly discriminative: in our setting lots of scenes contain a chair below a keyboard below a screen. This problem is also present in methods where low-resolution whole-images are used as descriptors, so we borrow the SeqSLAM technique developed for that domain [23] where localization is done over a contiguous sequence of images. The last N observations are compared with the memorized observations and a score table is recorded. Since the robot may have different velocities in the teach and repeat phases, we must test different velocities to find a good match. Figure 3.4 shows an example score table and line segments corresponding to different UAV velocities. We find the line that passes though score cells with the highest sum. The gradient of the line gives the UAV velocity and the peak value along the line gives the current position along the trajectory.

Although sequential matching helps the system to reduce the effect of perceptual aliasing, we can also benefit from the fact that matching has to be continuous in time. Since the taught path is recorded over a continuous period of time, we know that robot can not jump over in time. To address this, after initial relocalization, we positively weigh descriptors nearby in the sequence to add a temporality constraint. The more confident the previous match, the narrower is the distribution we weigh nearby descriptors with. Also, while testing relocalization, we found that in trajectories with repeated sections (time-extended loop-closures), basic sequence matching could match the wrong location and the robot could become stuck in a loop. With this mechanism, the robot can correctly repeat trajectories with repeated parts.

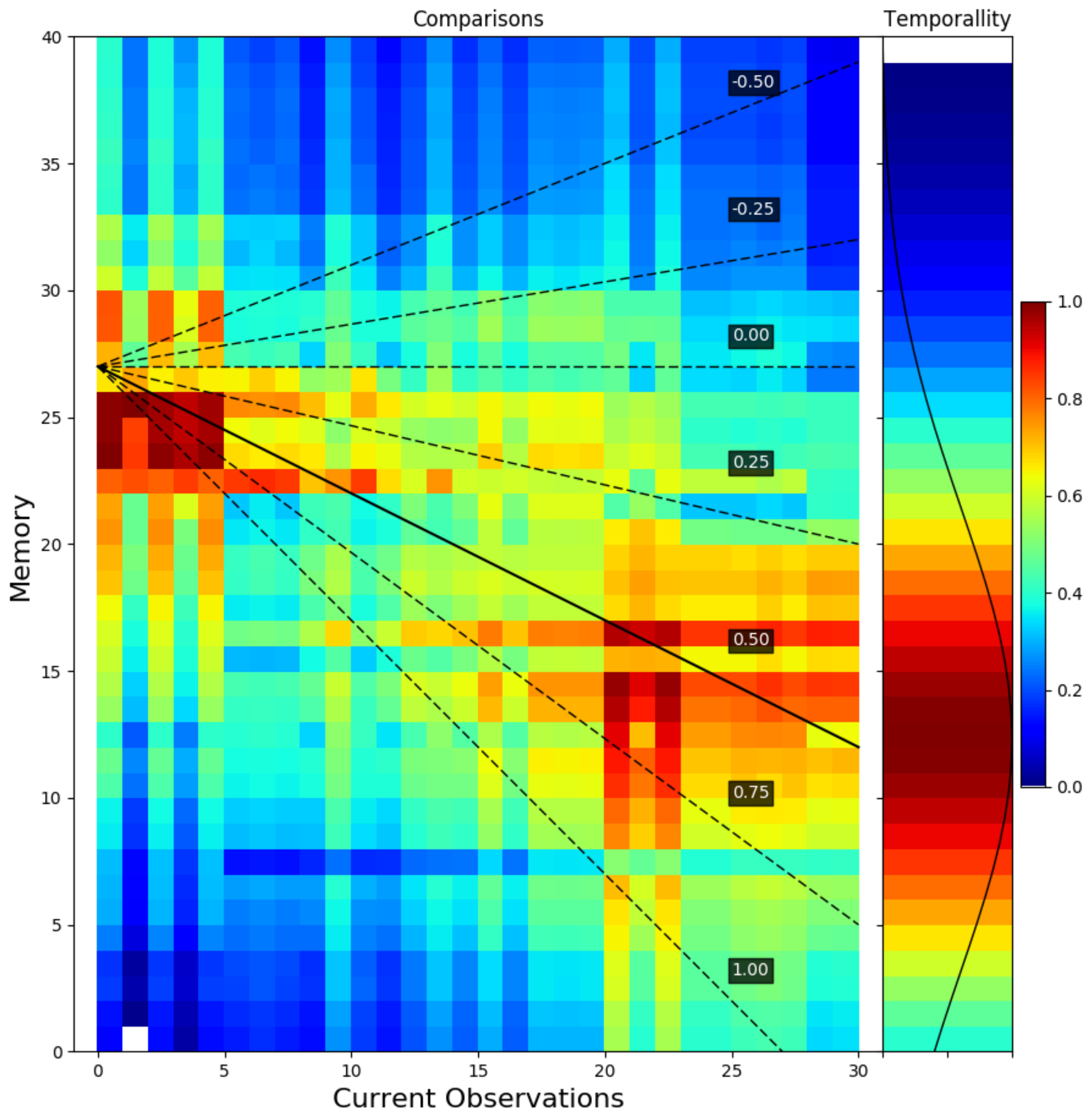

Figure 3.4: Extended SeqSLAM relocalization method: heatmap of the similarity of a sequence of 10 recent scene descriptors compared to memorized descriptors in reference memory. The line segment with the highest sum of values gives the best estimate of current UAV position and velocity. After initial relocalization we weight nearby scenes to impose a temporality constraint that avoids being confused by time-extended loop-closures. [44] (Copyright ©2018, IEEE)

Adding the temporality constraint, the similarity measure becomes:

$$S_t = \frac{S_s(1 + \beta \cdot \text{norm}(x = i, \mu = i', \gamma))}{(1 + \beta)}, \tag{3.3}$$

Where S_s is the score of matching that has been done by sequential comparison and β is the factor of temporality. β is selected by the confidence of previous match. i is the index of the memory which is being calculated and i' is the average of the last 5 matched indices of the memory.

3.2.5 Motion Control

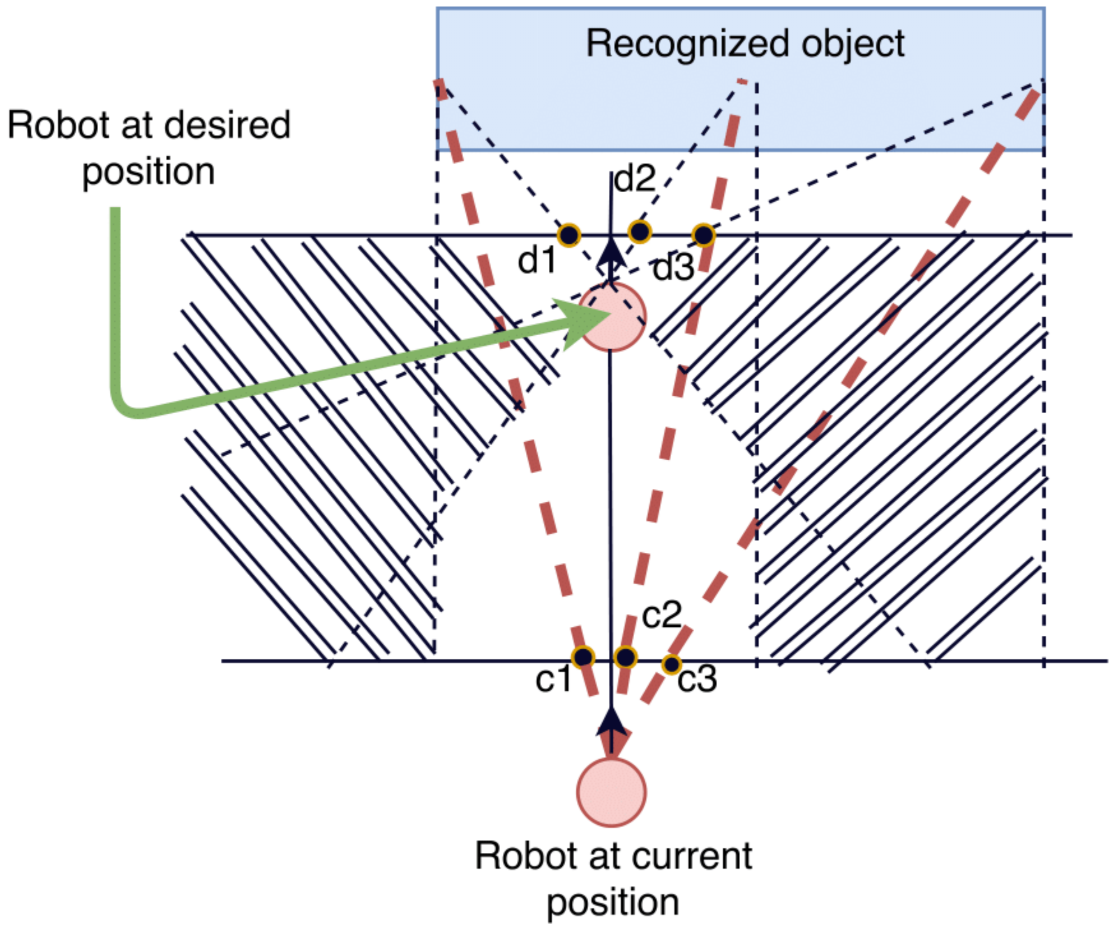

Figure 3.5: Schematic Funnel Controller adapted from [6]: the controller predicts action given the current robot position, desire position, and recognized objects. [44] (Copyright ©2018, IEEE)

Our learned trajectory is in the form of object locations within keyframes, so we use a following controller afforded by this information. The "Funnel lane" path following controller [6] method uses image-space point landmarks to achieve robust visual servoing. We extended it here for the case of object landmarks. We control robot pose on a 2D plane of constant altitude only, but the method is trivially extended to altitude control. Three points are considered for each object detection: one at the center and one at each extreme of its bounding box in the horizontal axis. Using the extent of the object allows us to reason about its scale change (but not its size in 3D). Figure 3.5 sketches the geometry of how two detections of the same object can be used to construct a “funnel lane’ that guides the motion of the robot to obtain the second camera position given the first.

One object detection is enough to repeat an (x, y) trajectory with respect to that object, but the approach direction is not constrained. A second object allows us to control yaw, to get replay of the full sequence of 2D poses (x, y, θ).

Equation 3.4 shows the funnel lane controller response for the single landmark on the left x-extent of the recognized object in Figure 3.5. c_1 and d_1 are the horizontal distance of the land mark in image space of the current position and desired position respectively, and $\phi(c_1, d_1) = \frac{1}{\sqrt{2}}(c_1 - d_1)$.

Equation 3.5 averages the response for a whole object over its three landmarks. The final response is a command for heading adjustment. We scale the output by a constant gain for tuning (we used a factor of 12).

$$\theta_{c_1} = \begin{cases} \gamma \min\{c_1, \phi(c_1, d_1)\}, & \text{if } c_1 > 0 \text{ and } c_1 > d_1, \\ \gamma \max\{c_1, \phi(c_1, d_1)\}, & \text{if } c_1 < 0 \text{ and } c_1 < d_1, \\ 0, & \text{otherwise.} \end{cases} \tag{3.4}$$

$$\theta_{\text{obj}} = \frac{(\theta_{c_1} + \theta_{c_2} + \theta_{c_3})}{3}. \tag{3.5}$$

Often, several objects are detected in a single scene. When this occurs, the controller calculates the response for each object individually using Equation 3.1. Then it performs the funnel lane method for all objects as sketched in 3.5. We can then compute the response for the entire image by averaging objects responses:

$$\theta_{\text{image}} = \frac{\sum_{n=1}^{N} \theta_{obj}}{N}. \tag{3.6}$$

We then perform Equation 3.6 on set of frame starting from the best matched to N future frames to align the robot with the best matched frame as well as looking ahead to align with upcoming frames:

$$\theta_{\text{Funnel}} = \frac{\sum_{i=I_m}^{W_f} \theta_{\text{image}}(\text{current}, i)}{W_f}. \tag{3.7}$$

Where I_m is the index of the matched keyframe and W_f denotes the size of funnel control window. We used $W_f = 30$.

The basic funnel controller has a failure mode that is difficult for our application: it steers the robot to align the currently-perceived objects as they should appear in later frames (in

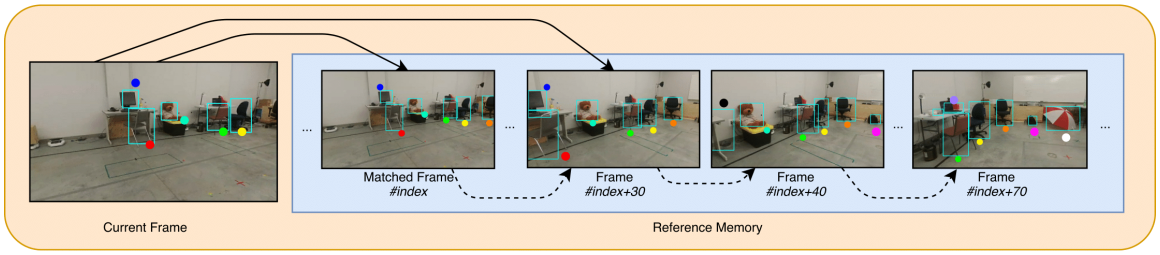

Figure 3.6: Funnel Lane with short look-ahead window plus Virtual Funnel Lane with long look-ahead window to react to unseen but predicted objects. [44] (Copyright ©2018, IEEE)

short look-ahead). But it does not react to upcoming objects that can not yet be seen. Thus with a narrow field of view camera it fails to turn the robot in sharp corners. To address this, we augment the funnel lane controller with what we will call a *virtual funnel lane*: we simulate the action of the normal funnel lane controller by running it on the sequence of stored images from the current best match up to some window look-ahead size. The robot simulates what will happen in the future, assuming it is correctly localized. This was the robot can “see around" upcoming corners. The robot records the heading changes prescribed by the controller in this virtual look-ahead flight, and averages them to obtain a predicted future yaw value as described in Equation 3.8. This is then blended with the original funnel lane control by Equation 3.9. The intuitive effect of these two control components is that the “real" funnel lane aligns the robot well to the things it can see now, while the virtual funnel lane pre-aligns the robot to objects it can not yet see. Thus the robot will turn nicely around corners even when the set of objects in view changes completely.

$$\theta_{\text{Virtual}} = \frac{\sum_{i=I_m}^{W_v} \theta_{\text{image}}(i, i+1)}{W_v} \tag{3.8}$$

Where I_m is the index of the matched keyframe like equation 3.7 and W_v denotes the size of the virtual funnel window. We used $W_v = 70$.

$$\theta_{\text{total}} = \alpha \cdot \theta_{\text{Funnel}} + (1 - \alpha) \cdot \theta_{\text{Virtual}} \tag{3.9}$$

Where θ_{Funnel} is the weight of funnel controller range between 0.0 to 1.0, that indicates the importance of funnel controller and virtual funnel controller respectively. We chose this by experiment, to get a working value of 0.7.

This enhanced controller is illustrated in Figure 3.6. The current image is only being compared with the next 30 key-frames in the memory to generate a funnel lane control command, but the virtual funnel lane looks ahead 70 key-frames to consider objects not yet in view such as the umbrella. The look-ahead anticipates the turn towards the umbrella and forces the controller to change the robot's heading towards it.

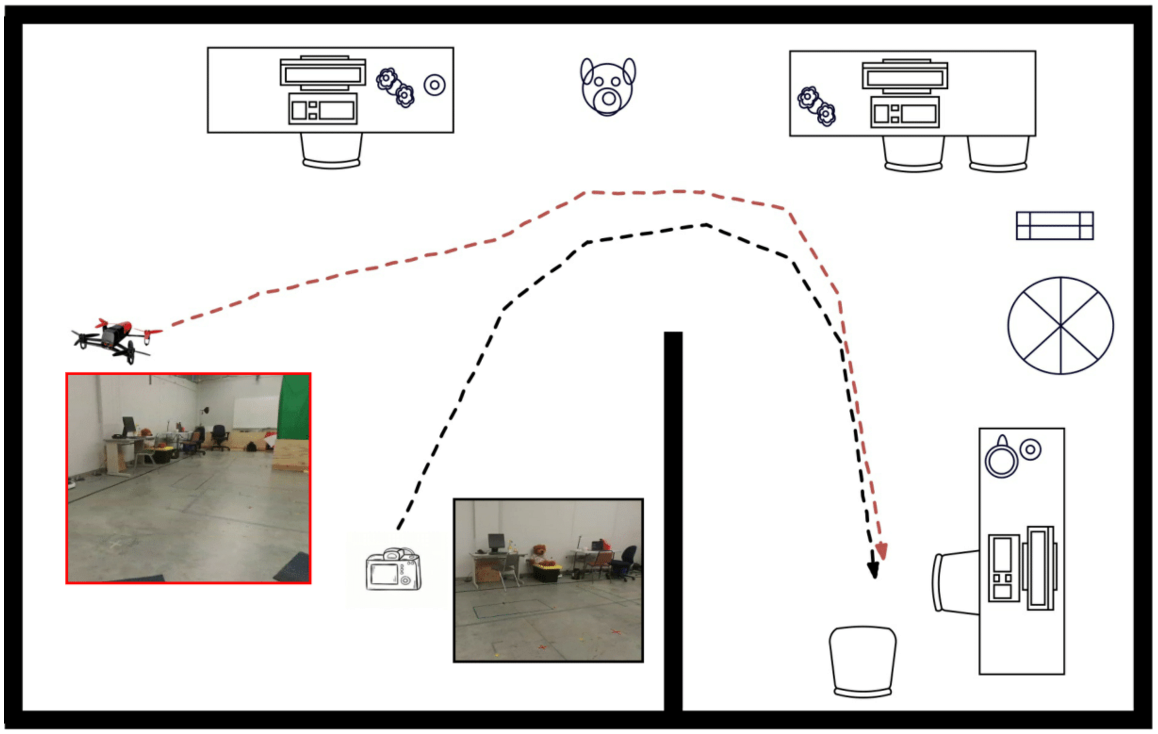

Figure 3.7: Test scenario: The trajectory of the camera is encoded as a series of object landmark locations in image space. Later, the UAV autonomously localizes itself in this space, then repeats the remaining part of the camera trajectory. A CNN detects objects such as desk, chair, mug, umbrella, backpack in keyframes. The UAV trajectory is generated directly from pairs of keyframes. [44] (Copyright ©2018, IEEE)

To complete, control logic is: (i) rotate on the spot if not well localized; (ii) if yaw error is below a threshold, go forward with constant velocity, else yaw to correct the error. We decoupled yaw and forward motion as the low-level flight controller behaved badly when attempting to yaw and translate at the same time due to the complex vehicle and controller dynamics of a quad-rotor.

3.3 Experiments

We performed real-world experiments with a commodity Bebop-2 UAV. We are limited to a 10m x 6m experimental arena where we can independently measure UAV trajectories using a Vicon motion-capture system. The test environment is pictured in schematic in Figure 3.7. We limited the total number of recognizable objects in an office-like environment to 18 including chairs, tables, monitors, and so on.

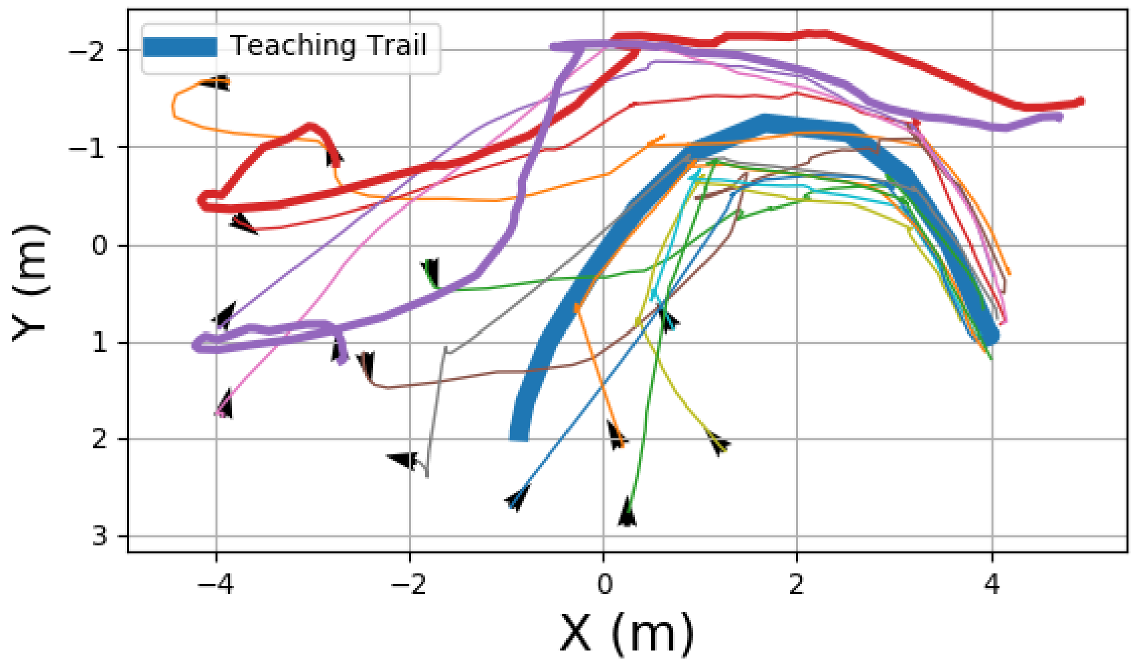

Figure 3.8: Trails in both teaching and repeating phases during experiment I, arrows are indicating initial rotation of the robot. [44] (Copyright ©2018, IEEE)

3.3.1 Teaching

In the training phase, the UAV is used as a passive camera. It is placed on a wheeled cart and pushed along the desired route, indicated by the trajectory marked with a camera Figure 3.7. The altitude is constant at 1.5m. The pose is recorded independently by motion capture for further analysis. The video is fed to the CNN object detector and descriptor generator and the sequence of memory of 290 key-frames is built.

3.3.2 Experiment I: Changing Viewpoint

A first experiment was performed to test the robustness of relocalization and trajectory repeating to changes in initial camera viewpoint. We run 12 trials with the the robot starting from (x, y, θ) poses chosen at random up to $5m$ away from the original taught location, and one of two fixed altitudes chosen at random. The robot flies autonomously, controlled by the VTR system. On startup, the robot rotates on spot until relocalization is achieved, then attempts to repeat the trajectory until matching the last keyframe.

Results

In 10 out of 12 trials, robot has completed the learned trajectory, arriving less than $0.5m$ from the original end-point. Figure 3.8 reports the start positions and the paths of all robots. The blue bold line is the taught path. Note that the robot navigates around an opaque wall

(Fig 3.7), so that the features detected in the second half of the trajectory are not visible in the first half: direct visual servo to the final target can not solve this task. In both the failed cases, the robots incorrectly detected the endpoint keyframe too early, stopping at the pair of chairs at the top right, and not the correct final pair of chairs at the bottom right. Although, sequentially and temporality constraints are added to overcome these perceptual aliasing problems, but in this case descriptors were similar in sequence and close in time. Both trails did correctly localize initially and completed 65% of the trial correctly.

3.3.3 Experiment II: Changing Environment

A second experiment investigates the ability to repeat a learned trajectory when the environment changes. The robot starts approximately at the same place as in teaching. But in one set of trials we change the lighting by turning off the ceiling lights and turn on two spot lamps. In a second set of trials we remove five objects that were noted by the object detector in training, and move eight others. The supplementary video shows us disturbing the objects to change their appearance. We repeat the trajectory and record the time taken to arrive at the final landmark.

Results

Examining robustness to appearance changes, we record the time taken to repeat the trajectory after (a) drastically changing the lighting and (b) removing objects and moving others to change their appearance. A histogram of flight times is shown in figure 3.10 with fitted Gaussians.

Removing and moving objects changes the execution time distribution, increasing it by an mean of 4sec (less than 10%).

But, perhaps surprisingly, changing the illumination reduces the time taken for a repeat. The data show that the changed illumination slightly reduced the number of recognized objects, in particular the objects that were less repeatably detected in the teaching phase. Without objects disappearing and reappearing as frequently, the flight controller produced smoother behaviour and reached the goal faster. Perhaps the method could be improved by deliberately filtering such “flickering" objects.

Figure 3.9 provides evidence of this effect, showing that the number of detected objects differs during the light-changing and removing/moving objects experiments. When removing/moving objects, the number of detected features decreases, but the run time is longer because the moved objects degrade the controller behaviour, and the robot takes longer to line itself up along the trajectory. In the spot-lighting trials, the fewer object detections improve completion time as described above.

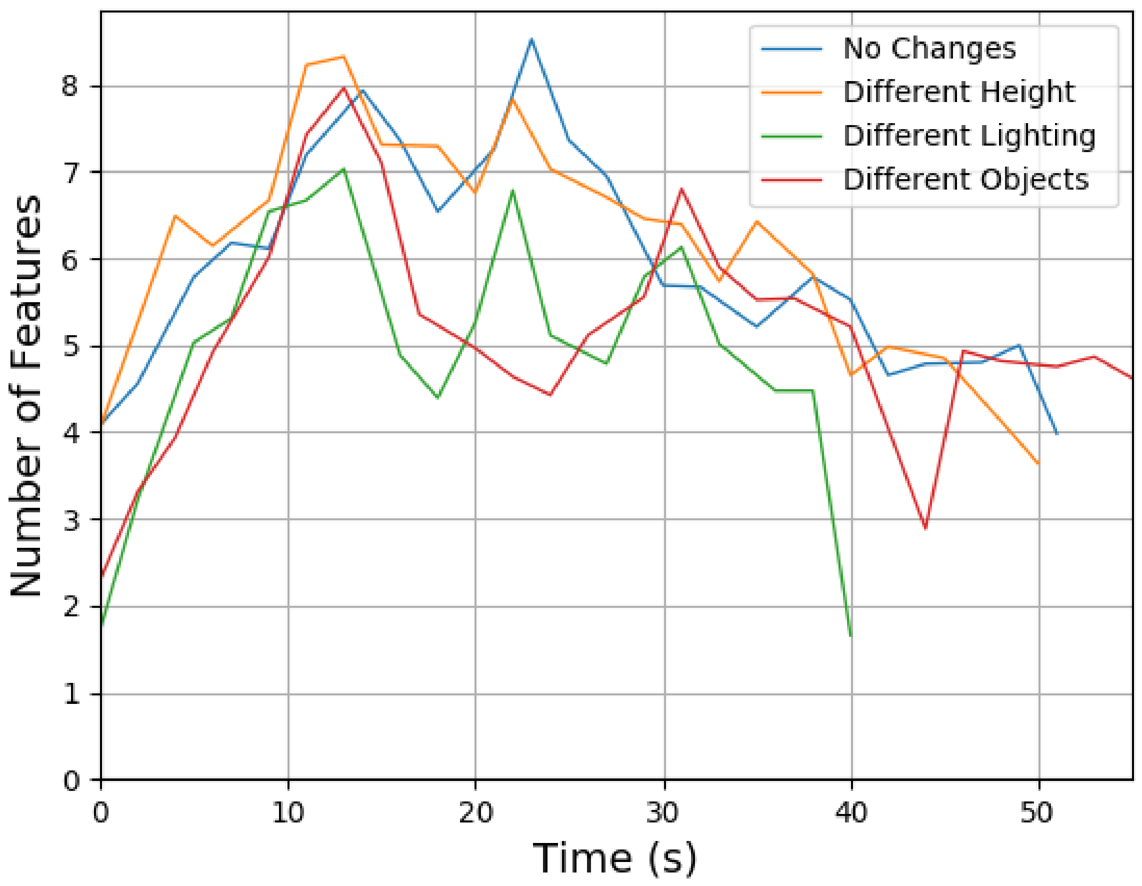

Figure 3.9: Number of features detected at nearest-corresponding locations during teaching and repeat phases. [44] (Copyright ©2018, IEEE)

3.4 Conclusion, limitations and future work

The work presented in this chapter showed that a current deep-CNN object detector can be used to provide robust and repeatable features that are sufficient for monocular VTR. We demonstrated the method with real UAV experiments, but in an lab/office-like environment as a substitute for our eventual goal of large scale outdoor flights. The indoor environment allowed us to use a pre-trained object detector since a detector for UAV navigation tasks is not available, and allowed us to collect ground truth data from an installed motion capture system. Objects are detected with viewpoint and lighting invariance that compares favourably with other methods, and with interesting novel invariances to the object being completely turned around or replaced by another of the same category. We demonstrated that we can start the robot 5m away from the taught initial position, facing at a random heading, and the robot could relocalize itself and complete the trajectory using the object detections alone. Comparable experiments using SURF features have been shown to work at not more than 1m disturbance. We also successfully repeated a trajectory with the UAV's

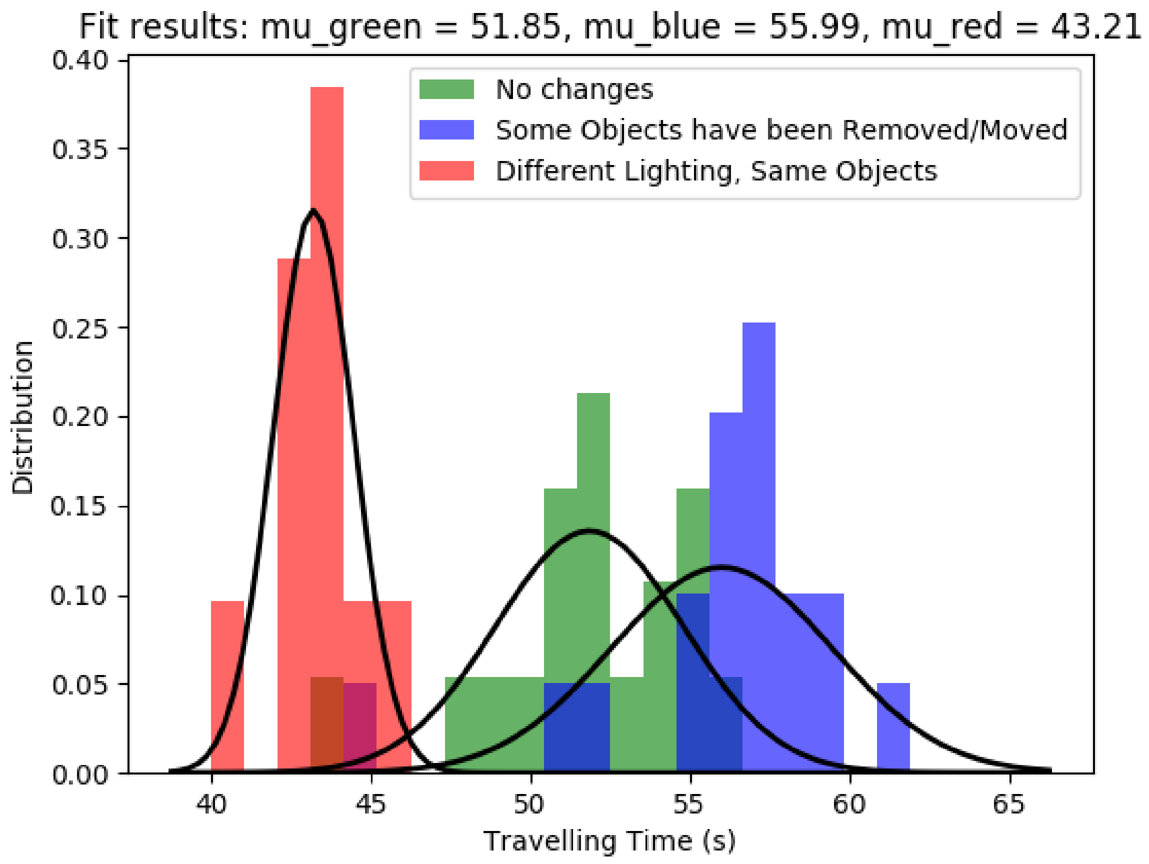

Figure 3.10: Repeat completion times for identical environment (green), light-changed environment (red) and object-moved environment (blue). [44] (Copyright ©2018, IEEE)

altitude changed by 40%, and also if several objects are removed and removed, or if the global lighting is changed dramatically.

We also contribute a novel two-window use of the funnel lane visual servo method that looks ahead in time to respond to upcoming objects.

Our method depends on a supply of distinct ambient objects. It will not work in environments without discontinuous, recognizable object categories. Previous work on automatic generation of low-level feature dictionaries could suggest directions for future work. For taking this work outdoors over kilometer scales, we need a CNN that reliably detects objects that exist in our target environment. We may have to train our own networks on the salient objects. In settings where distinct objects are rare, we could extend the method to use point features and/or inertial sensing to allow navigation in object-free areas.

One simple but interesting extension of the work would be for the robot to announce changes to the set of objects. If a previously-seen vase or painting is missing, perhaps it has been stolen. If a backpack has appeared, perhaps it was left accidentally. More substantially, perhaps a robot is performing an inspection trajectory over a new manufactured object such as an aeroplane wing. Is a rivet missing or has a bad weld appeared, compared to the teach phase recorded over a known-good example? If patrolling a construction site at night, a newly arrived truck or absent machine could be a security issue.

The recent performance improvement in some computer vision tasks due to CNNs and similar methods are giving us interesting new choices for robot vision. We aim to exploit the new robust vision methods towards more complete robot autonomy and new applications.

Chapter 4

Conclusion

In this thesis, we had two main contributions. The first is an introduction of a new method for semi-autonomously control of drones which is based on object tracking and visual servoing, and the second is the development of semantic-based visual teach and repeat.

In chapter 2, we showed that our presented semi-autonomous approach can help pilots control a drone towards very long-distance targets by only selecting a few bounding boxes. We have tested our method in the real world to approach a dog with 30cm height starting from $150m$ away by only selecting a few targets in this path. We enabled users to select targets in the image transmitted from the drone while the drone controls its height, pose and, velocity to reach the target.

Our monocular visual servoing technique estimates the distance to the target using the prior on the height of the target. We use the distance from the target and pitch angle of the camera to calculate the relevant pose of the object in the drone's frame and tune parameters to control the drone towards the target.

By considering the minimum required size of the target in an image to have robust tracking, we have introduced a track-ability locus that represents the area around a target which tracker can be initiated and successfully track the target. We have tested this in the simulation by running a tracker on the simulated model of a drone and by selecting a building and a window with the height of 11m and $2m$ accordingly as targets. We showed that if the initial point of the drone was inside the track-ability locus of the targets, it could successfully reach the final target (window in this case) from the distance over $190m$ by only the minimum attention required from the pilot (selecting building and window as the targets).

In chapter 3, we have introduced an object-based monocular visual teach and repeat. We showed that a deep-CNN object detector can be used to provide robust and repeatable features that are sufficient for repeating a traversed path using only a monocular camera as a sensor. We demonstrated the method with real UAV experiments in a lab/office-like environment. We demonstrated that we can start the robot 5m away from the initial path and facing it at a random heading, and the robot still could relocalize itself and complete the trajectory. We showed that our algorithm is robust to the lighting condition and changes in pose or removal of few objects from the scene.

Our proposed algorithm saves a lot of memory compared to old-fashioned methods which rely on hand-crafted features. We found that each scene might contain two to eight objects in our experiment setup, so the resulting whole-scene descriptor is small: of the order of 160 bytes per image, while a single original SIFT point descriptor is 512 bytes.

We had contributed to the funnel lane visual servo method by looking ahead in time to respond to upcoming objects and, we also extend the SeqSLAM relocalization method to work with object features.

Bibliography

[1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[2] Sean L Bowman, Nikolay Atanasov, Kostas Daniilidis, and George J Pappas. Probabilistic data association for semantic slam. In *International Conference on Robotics and Automation (ICRA)*, pages 1722–1729. IEEE, 2017.

[3] Drone Delivery Canada. Reimagining the way you deliver. `https://dronedeliverycanada.com/technology/`, 2021.

[4] Jessica R Cauchard, Alex Tamkin, Cheng Yao Wang, Luke Vink, Michelle Park, Tommy Fang, and James A Landay. drone. io: a gestural and visual interface for human-drone interaction. In *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 153–162. IEEE, 2019.

[5] François Chaumette and Seth Hutchinson. Visual servo control. i. basic approaches. *IEEE Robotics & Automation Magazine*, 13(4):82–90, 2006.

[6] Zhichao Chen and Stanley T. Birchfield. Qualitative vision-based path following. *IEEE Transactions on Robotics*, 25(3):749–754, 2009.

[7] Lee Clement, Jonathan Kelly, and Timothy D Barfoot. Robust monocular visual teach and repeat aided by local ground planarity and color-constant imagery. *Journal of Field Robotics*, 34(1):74–97, 2017.

[8] Frsky. Tranis q x75 remote control. `https://www.frsky-rc.com/product/taranis-q-x7s-access/`, 2021.

[9] Markus Funk. Human-drone interaction: let's get ready for flying user interfaces! *Interactions*, 25(3):78–81, 2018.

[10] Paul Furgale and Timothy D Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 27(5):534–560, 2010.

[11] Eberhard Graether and Florian Mueller. Joggobot: a flying robot as jogging companion. In *CHI'12 Extended Abstracts on Human Factors in Computing Systems*, pages 1063–1066. 2012.

[12] Klaus Haag, Sergiu Dotenco, and Florian Gallwitz. Correlation filter based visual trackers for person pursuit using a low-cost quadrotor. In *2015 15th International Conference on Innovations for Community Services (I4CS)*, pages 1–8. IEEE, 2015.

[13] Trevor Hall. Dji draw: How and when to use this flight mode. https://www.letusdrone.com/dji-draw-how-and-when-to-use-this-flight-mode/, 2021.

[14] DJI Inc. Dji official website for canada. https://www.dji.com/ca, 2021.

[15] Skydio Inc. Skydio 2 and x2. https://skydio.com/, 2021.

[16] This Is Engineering Inc. Shift, next generation of drone controlling system. https://www.thisiseng.com/drones/shift-red/, 2021.

[17] Hao Kang, Haoxiang Li, Jianming Zhang, Xin Lu, and Bedrich Benes. Flycam: Multitouch gesture controlled drone gimbal photography. *IEEE Robotics and Automation Letters*, 3(4):3717–3724, 2018.

[18] Tomáš Krajník, Pablo Cristóforis, Keerthy Kusumam, Peer Neubert, and Tom Duckett. Image features for visual teach-and-repeat navigation in changing environments. *Robotics and Autonomous Systems*, 88:127–141, 2017.

[19] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision (ECCV)*, pages 21–37. Springer, 2016.

[20] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[21] Kirk MacTavish, Michael Paton, and Timothy D Barfoot. Visual triage: A bag-of-words experience selector for long-term visual route following. In *International Conference on Robotics and Automation (ICRA)*, pages 2065–2072. IEEE, 2017.

[22] Matt McFarland. Google drones will deliver chipotle burritos at virginia tech. *CNN Money, September*, 2016.

[23] Michael J Milford and Gordon F Wyeth. Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In *International Conference on Robotics and Automation (ICRA)*, pages 1643–1649. IEEE, 2012.

[24] Sepehr MohaimenianPour and Richard Vaughan. Hands and faces, fast: mono-camera user detection robust enough to directly control a uav in flight. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5224–5231. IEEE, 2018.

[25] Mani Monajjemi. bebop autonomy, ros driver for parrot bebop drone 1.0 and 2.0. https://bebop-autonomy.readthedocs.io/en/latest/, 2021.

[26] Mani Monajjemi, Sepehr Mohaimenianpour, and Richard Vaughan. Uav, come to me: End-to-end, multi-scale situated hri with an uninstrumented human and a distant uav. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4410–4417. IEEE, 2016.

[27] Valiallah Monajjemi, Shokoofeh Pourmehr, Seyed Abbas Sadat, Fei Zhan, Jens Wawerla, Greg Mori, and Richard Vaughan. Integrating multi-modal interfaces to command uavs. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*, pages 106–106, 2014.

[28] Beipeng Mu, Shih-Yuan Liu, Liam Paull, John Leonard, and Jonathan P How. Slam with objects using a nonparametric pose graph. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 4602–4609. IEEE, 2016.

[29] Trung Nguyen, George KI Mann, Raymond G Gosine, and Andrew Vardy. Appearance-based visual-teach-and-repeat navigation technique for micro aerial vehicle. *Journal of Intelligent and Robotic Systems*, 84(1-4):217–240, 2016.

[30] IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS Vancouver 2017. `https://www.iros2017.org/`, 2017.

[31] Michael Paton, Kirk MacTavish, Michael Warren, and Timothy D Barfoot. Bridging the appearance gap: Multi-experience localization for long-term visual teach and repeat. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1918–1925. IEEE, 2016.

[32] Andreas Pfrunder, Angela P Schoellig, and Timothy D Barfoot. A proof-of-concept demonstration of visual teach and repeat on a quadrocopter using an altitude sensor and a monocular camera. In *Canadian Conference on Computer and Robot Vision (CRV)*, pages 238–245. IEEE, 2014.

[33] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525. IEEE Computer Society, 2017.

[34] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 91–99, 2015.

[35] John G Rogers, Alexander JB Trevor, Carlos Nieto-Granda, and Henrik I Christensen. Simultaneous localization and mapping with learned object recognition and semantic data association. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1264–1270. IEEE, 2011.

[36] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[37] Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1352–1359, 2013.

[38] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, pages 621–635. Springer, 2018.

[39] Giuseppe Silano, Pasquale Oppido, and Luigi Iannelli. Software-in-the-loop simulation for improving flight control system design: a quadrotor case study. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 466–471. IEEE, 2019.

[40] Niko Sünderhauf, Feras Dayoub, Sean McMahon, Ben Talbot, Ruth Schulz, Peter Corke, Gordon Wyeth, Ben Upcroft, and Michael Milford. Place categorization and semantic mapping on a mobile robot. In *International Conference on Robotics and Automation (ICRA)*, pages 5729–5736. IEEE, 2016.

[41] J. Surber, L. Teixeira, and M. Chli. Robust visual-inertial localization with weak gps priors for repetitive uav flights. In *International Conference on Robotics and Automation (ICRA)*, pages 6300–6306, May 2017.

[42] Tsukamoto Taisho and Tanaka Kanji. Mining dcnn landmarks for long-term visual slam. In *International Conference on Robotics and Biomimetics (ROBIO)*, pages 570–576. IEEE, 2016.

[43] Dante Tezza and Marvin Andujar. The state-of-the-art of human–drone interaction: A survey. *IEEE Access*, 7:167438–167454, 2019.

[44] Amirmasoud Ghasemi Toudeshki, Faraz Shamshirdar, and Richard Vaughan. Robust UAV visual teach and repeat using only sparse semantic object features. In *Canadian Conference on Computer and Robot Vision (CRV)*, pages 182–189. IEEE, 2018.

[45] Wepulsit. Wepulsit, intuitive drone control with only one hand. `http://www.wepulsit.com`, 2021.

[46] Scott Niekum. ROS wiki: ar-track alvar. `http://wiki.ros.org/ar_track_alvar`, 2021.

[47] Chao Yao. Gazebo models and worlds collection. `https://github.com/chaolmu/gazebo_models_worlds_collection`, 2021.