Any Object Tracking and Following by a Flying Drone

Roman Barták and Adam Vyškovský
Charles University in Prague, Faculty of Mathematics and Physics
Malostranské náměstí 25, Praha 1, Czech Republic
e-mail: bartak@ktiml.mff.cuni.cz

***Abstract*—AR.Drone is a quadcopter with onboard sensors including a frontal camera. The drone can be controlled from a computer via WiFi. This paper describes a method for autonomous tracking of a selected object by AR.Drone. We utilize a computer-vision approach called tracking-learning-detection (TLD) to track an arbitrary object selected by a user in the video-stream going from the front camera of the drone. Information about location of the tracked object is then used to guide the drone using the proportional-integral-derivative (PID) controller. The method was implemented in software FollowMe.**

I. Introduction

Recent development in robotics brought inexpensive robots, such as AR.Drone, with various sensors and actuators. Many tasks solved by these robots became popular in recent years including object tracking and following. In this paper we show that even inexpensive drones can do autonomous object tracking and following. The idea is that in a snapshot taken by the drone's front camera, the user marks any area as an object of interest and then the drone autonomously follows this selected object as the object moves. It is supposed that the drone behaves in real environment with possibly changing lighting conditions and that the object of interest changes slightly its appearance as the drone sees the object from different angles. Moreover, the software is supposed to control the drone in a dynamic environment with possible external forces, e.g., small turbulences etc. Only the built-in sensors and actuators of the AR.Drone will be used, though due to limited computing power of the built-in processor we will use an external computer (a mainstream laptop) to do reasoning.

There exist many methods for detecting and tracking objects in a video stream; Visual Object Tracking challenge (http://www.votchallenge.net/) gives a good overview of existing methods. We tried several classical methods such as template matching, color detection, feature detection and matching, and motion estimation [7] but they all cover only specific aspects of the object of interest. This is also the case of the method of color based thresholding used in [1] to solve a similar task. To realize the presented goal of any object tracking we exploited a novel object tracking technique called *tracking-learning-detection* (TLD) [2]. This method combines advantages of object tracking and detection which is critical because the object of interest can easily disappear from drone's view (hence we need object detection) while the tracking method can capture different views of the object that can be used to teach the detector. TLD is also used in a similar project [6]. As the size of the target object is unknown we added a method to estimate scale and hence distance from the drone. The output of TLD, which is a position of the object of interest in the video frame, together with its distance is then fed to a classical *proportional-integral-derivative* (PID) controller [3], that tells the drone where to fly.

This paper shows how the above-mentioned techniques work together to solve the problem of any object tracking and following by a flying drone. We will first describe the robotic platform used – AR.Drone 2.0 – to show its capabilities which drive further development. After that we will give technical details of the tracking method, namely the tracking algorithm, the detection algorithm, their integration, and bootstrapping. Then we will talk about the controller and about the implementation and user interface for the software.

II. Parrot AR.Drone Platform

AR.Drone 2.0 by Parrot Inc. is a high-tech flying toy that can be used for augmented-reality games. Technically, it is a quadcopter with sensors and a controller. AR.Drone is equipped with two cameras, one facing down and one forward. The bottom camera is used by the onboard software to stabilize the drone by preventing drift. The front camera, used in our software, has a resolution 1280×720 pixels running at 30 fps. The drone is further equipped with a 3-axis gyroscope measuring pitch, roll, and yaw (Figure 1), and a magnetometer that increases accuracy of these measurements. There is also a 3-axis accelerometer, which is able to measure acceleration in all three dimensions. Altitude is measured by an ultrasound sensor and a pressure sensor. The ultrasound sensor measures altitude just above the ground (heavily exploited in the takeoff and landing). The pressure sensor aids in measuring altitude several feets above the ground where the ultrasound sensor does not give reasonable estimates. Precision of this sensor is up to 10 Pa, which corresponds to 80 cm.

AR.Drone is controlled by a 1GHz 32 bit ARM Cortex A8 processor with 1GBit of DDR2 RAM running at 200MHz. This processing architecture controls the basic operations of the drone including stabilization. The system is running a GNU/Linux so it is possible to install own programs there but due to a limited computing power we have decided to use an external computer.

978-1-5090-0323-5/15 $31.00 © 2015 IEEE
DOI 10.1109/MICAI.2015.12

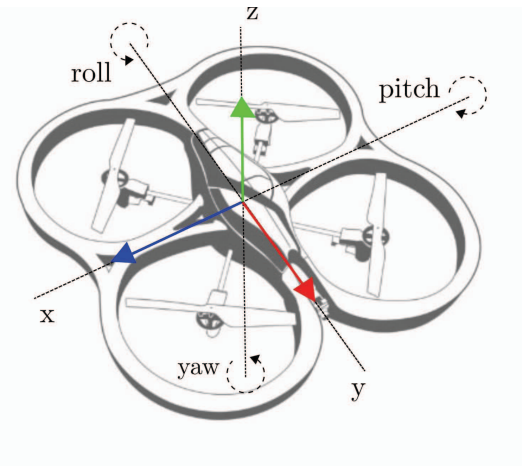

Fig. 1. AR.Drone is controlled by setting roll, yaw, and pitch.

The drone can be controlled externally via WLAN through a set of commands sent from the ground to the quadcopter. All the communication is done using three UDP channels. A Command channel allows sending commands to the drone. The device accepts the following commands with frequency 30Hz: takeoff, land, set limits (maximal declination, speed, and altitude), calibrate sensors, swap cameras (see below), set rotation speeds of rotors, set pitch, roll, yaw, and vertical speed. A NavData channel provides information about the drone state again with frequency 30Hz. The following information is provided: drone state (flying, steady, landing, takeoff, calibration, booting), sensor data (current pitch, roll, yaw, altitude, battery level, and speed in all axes). Finally, a Stream channel provides visual information from the cameras. The stream continuously transmits either the front view or the bottom view. Switching between the views is simply a matter of sending a particular command to the quadcopter indicating which view to activate. For our purposes it is quite natural to focus on the front camera. The video stream is encoded with a proprietary format, which is fortunately similar to the H.264 codec and can be decoded using standard software packages such as OpenCV without any additional intervention. However, one of camera's downsides is the fact that the images are subject to significant distortions and especially while in flight the images are heavily blurred.

III. Object Detection and Tracking

The critical part of target tracking is detection of tracked object in a video stream captured by drone's front camera. The exact location of target is then used to navigate the drone as we will describe in the next section. Formally, the problem can be stated as follows. We have a finite sequence of matrices $(M_0, M_1, \ldots, M_n)$, where each matrix represents one frame of a video stream going from drone's camera. Entries in a matrix correspond to pixels in the picture. A set of pixels is selected in M_0 to identify the object of interest – target. We use a rectangular area to frame the target (see Figure 10). The task is to identify the target – the rectangular area – in each image M_i or to say that the target is not there. The detection system needs to assume that target's appearance may change as the target and the drone move. Nevertheless, such a change is typically not sudden.

There are basically two approaches to follow the target in a video stream. One approach uses a known location of target in a single video frame and by estimating motion between successive video frames the method computes the next position of the target. This is a natural method of object tracking that does not require a lot of information about the target as it can use general picture processing techniques such as optical flow [7]. On the other side, such approaches tend to accumulate error and hence their precision deteriorates with time as they start to drift away from the real trajectory of the tracked object. Moreover, when the target is lost the tracking methods cannot re-discover it again. The other approach is based on detecting the object of interest in a static picture by techniques such as template matching and feature detection and matching. The advantage of these approaches is that whenever the object appears in the picture they can detect it. The disadvantage is that the methods require extensive training to learn how the target looks and they are prone to change of appearance of the target.

We decided to use a novel approach to object tracking in a video stream that combines advantages of above-described object tracking and object detection. This method is called *tracking-learning-detection* (TLD) and it was introduced in [2]. The method decomposes the task of long-term object tracking into three subtasks: tracking, learning, and detection, hence the name TLD. The method accepts the fact that tracking and detection become error-prone when operating on their own, but they could be combined to form a pair that supports each other. The tracker can provide the detector with learning data real-time and the detector can reinitialize the tracker in case the object gets lost. Finally, the learner estimates the errors made by the detector and updates its model to avoid these errors in the future.

A. Tracking

Tracking is the process of estimating the motion of an object in consecutive video frames under the prerequisite that the position of the object was known in the previous frames. Recall that the user frames the object of interest by a rectangle in the first image so we have the initial position. The task is to identify the rectangle framing the target in next images while assuming that the rectangle scale may change as the target and drone move. The TLD algorithm uses the Lucas-Kanade tracker [4] enhanced by a forward-backward consistency checking to improve quality of tracking over a sequence of frames and also to self-evaluate the quality of tracking.

The Lucas-Kanade tracking algorithm [4] is based on optical flow, which estimates motion between two subsequent pictures. In general, optical flow methods use two assumptions. The first assumption is that the projection of the same point in

the real world on the image plane is the same in every frame (*the brightness constancy assumption*). The second assumption is that neighboring pixels show similar motion (*the spatial coherence constraint*). Let $I(x, y, t)$ be the intensity of a pixel at position (x, y) at a given frame in time t. Assume now that a pixel at position (x, y) moved by the vector (u, v) so its position in the next frame is $(x + u, y + v)$. The brightness constancy assumption tells us that $I(x, y, t) = I(x + u, y + v, t + 1)$ in two consecutive frames. Obviously this single equation is not enough to find the vector (u, v) as this is an under-constrained system. This is where the spatial coherence constraint is used. We take several pixels around the pixel of interest, in our case we took two pixels on the right, left, down, and up so in total 25 pixels, and the constraint says that all these pixels moved in the same way, that is, by the vector (u, v). Now, the system of equations is over-constrained so we are looking for vector (u, v) minimizing:

$$\sum_{-2\leq i,j\leq 2} (I(x + i, y + j, t) - I(x + i + u, y + j + v, t + 1))^2.$$

The Lucas-Kanade tracker uses a differential method to find the motion vector (u, v) for pre-selected key points and it also uses the method of optical pyramids and estimates the motion of objects from coarser to finer grain to speed up the algorithm. Figure 2 shows an example of optical flow found by the Lucas-Kanade method.

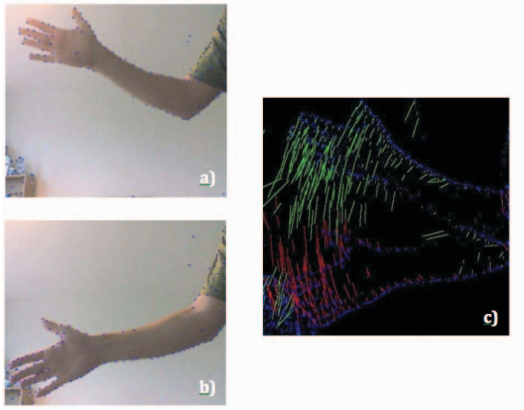

Fig. 2. Motion vectors computed by the Lucas-Kanade optical flow method: a) position of hand with key points before, b) position of hand after, c) computed optical flow.

When we know the movement of individual pixels we need to estimate the movement of the rectangle framing the target (*the bounding box*). It is also good to know the accuracy of this estimate. This is where the TLD uses *the forward-backward consistency checking* algorithm. This algorithm works as follows (see Figure 3). We take some k consecutive frames (two frames in our case) $(M_t, \ldots, M_{t+k})$ starting at time t and in the first frame M_t we select a point p_t. Then we apply a tracking algorithm that hopefully tracks the point along the k selected images and outputs a forward trajectory $T_F = (p_t, \ldots, p_{t+k})$ of this point. Afterwards, we repeat this exact same procedure, except that we apply it on the k selected images in the reverse order $(M_{t+k}, \ldots, M_t)$ with position $\hat{p}_{t+k} = p_{t+k}$ as the initial point. This will give us some backward trajectory $T_B = (\hat{p}_{t+k}, \ldots, \hat{p}_t)$. Finally, we take the T_F and T_B trajectories and compute the resulting forward-backward error, which is the (Euclidean) distance between p_t and $\hat{p}_t$. The smaller the distance is, the greater the confidence we have in the tracking algorithm that it was able to track the point correctly.

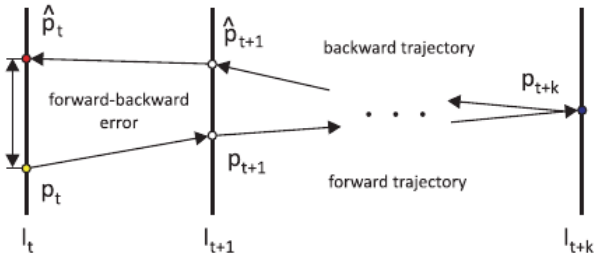

Fig. 3. Forward-backward consistency check method.

We apply the forward-backward consistency checking method in the following way. Inside the bounding box we select points on a rectangular grid (created by considering all intersections of ten horizontal and ten vertical equidistant lines inside the bounding box) and independently track these points by the Lucas-Kanade tracker. Then we perform the forward-backward consistency check on these points independently and assign a forward-backward error to each point. Half of the points with the largest error are filtered out. The remaining points then estimate the movement and scale of the bounding box. Figure 4 shows the tracked points in the initial bounding box and the preserved points in the final image. Basically, we compute a median of movement of the selected points both in the x and y coordinates to compute translation of the rectangle. Similarly we compute the scale of the bounding box by taking all pairs of tracked points, computing the ratio between the initial points distance and the final points distance in two consecutive images, and taking the median over all these ratios. This value determines the scale of the bounding box with respect to the original bounding box.

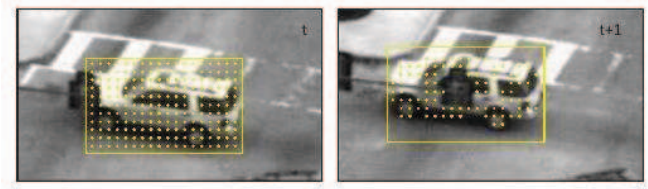

Fig. 4. Tracking single points independently inside the bounding box is resistant to partial occlusion.

B. Detection

We are interested in long-term object tracking and because the tracking algorithm may lose track of an object it is important to incorporate a detection mechanism in the long-term tracking algorithm as well. Detection is the process of finding an object in a single frame. The TLD algorithm incorporates a *scalable scanning-windows detector*. It creates a scanning-window of several scales and with each scale it slides along the image. Regardless of its initial size each scanning-window is scaled into a patch, which is a 15×15 square of pixels. For each patch the algorithm decides whether it contains the target or not. This is basically done by a classification algorithm that classifies the patch against a database of templates. The initial database of templates consists of the selected bounding box and a collection of boxes that have large overlap with the selected bounding box. Furthermore, these overlapping boxes are rotated, scaled and blurred, creating about 200 bounding boxes in total representing positive examples. Then a collection of some other random bounding boxes forming the background of the initial image are selected (these are the negative examples). All these boxes are scaled into patches. The database of templates is continuously updated as we will describe later.

The classification process has to be very efficient, therefore the algorithm cascades three stages of classifiers. In the first stage there is a patch variance classifier; in the second stage there is an ensemble classifier and in the third stage there is a nearest neighbor classifier (also called an NN-classifier). The *patch variance classifier* filters out bad patches (e.g. sky, walls) early and with little computation cost. We compute the gray-value variance of the patch with the object of interest from the initial image and reject patches whose variance is smaller than 50% of the variance of the initial patch. The *ensemble classifier* is composed of several base classifiers (ten in our case). The ensemble classifier decides about acceptance of a patch based on majority voting of base classifiers. We use base classifiers of the same type that work as follows. Each patch is modeled using a Boolean vector of size d (TLD uses $d = 13$), let us call it a *patch vector*. For a given classifier we use randomly selected d pairs of points in the patch. The pairs are restricted to have the same either horizontal or vertical coordinates, see Figure 5. If the pixels in the pair have similar intensities, then the pair is represented by value one in the patch vector, otherwise value zero is used. Each base classifier uses a posterior table of probabilities with 2^d possible indices describing all possible patch vectors. For each line in the table, we have a probability of a corresponding patch vector to represent a window containing the target. If this probability is at least 0.5 then the patch is classified positively as containing the target. The positively classified patches go to the last stage – the NN-classifier. We use several recent positive and negative patches and the NN-classifier looks for the past patch that is closest to the currently examined patch. Similarity between two patches p_1 and p_2 is computed by the formula:

$$S(p_1, p_2) = \frac{1}{2}(NCC(p_1, p_2) + 1),$$

where NCC denotes a *normalized correlation coefficient* from image processing. NCC is a real value between -1 and 1, therefore, the similarity measure between two patches ranges between zero and one (zero means two patches are completely anti-correlated and one corresponds to a perfect match). Let p^+ be the closest positive neighbor to patch p and p^- be the closest negative neighbor. Then we evaluate the patch p using the following similarity formula:

$$S^r(p) = \frac{S(p, p^+)}{S(p, p^+) + S(p, p^-)}.$$

$S^r(p)$ ranges between zero and one. If $S^r(p) > \theta$ then we treat the patch p as a positive patch, where θ is a parameter defining the decision boundary between negative and positive patches.

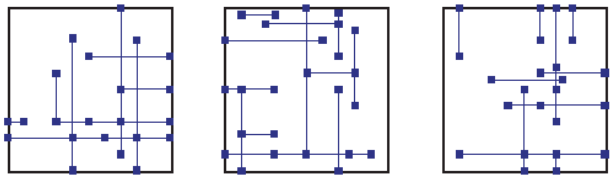

Fig. 5. Pixel comparisons measured within individual base classifiers. The three squares correspond to three different base classifiers. Blue lines join pixels (small blue squares) that are compared in these base classifiers.

In summary, for a patch to be classified as containing the target, the patch needs to pass through the sieve of three classifiers. The advantage of this approach is its scalability. The final classification of the patch is used to update the posterior probability tables of base classifiers in the ensemble classifier as well as to update the set of recent patches used in the NN-classifier – the positively classified patch increases the probability of its corresponding patch vector in each base classifier while the negatively classified patch decreases the probability.

C. Integration and Learning

Detection and tracking are running in parallel. We may obtain several bounding boxes from the detector (each p such that $S^r(p) > \theta$) and one bounding box from the tracker. The integrator selects the bounding box p with the largest value of $S^c(p)$, where $S^c(p)$ is computed similarly to $S^r(p)$ but assuming only the first 50% of positive patches. It may also happen that neither the tracker nor the detector finds a positive bounding box then the target is declared lost.

Initially the detector uses examples taken from the very first frame. However, the appearance of the target may change as the target and the drone move. Hence it is important to bootstrap the detector with new examples. This is where the TLD algorithm uses a new method called *P-N learning*. The

idea is to use two experts, P-expert focusing on positive examples and hence identifying false negatives and N-expert focusing on negative examples and detecting false positives. The P-expert assumes that the object moves on a trajectory. We can take several past positive boxes and the area of patches similar to these positive boxes (using $S^c(p)$ but with a larger θ) forms a so called *core*. If the box provided by the integrator is within this core, it is treated as reliable. Then other boxes around this reliable box are added as positive examples to the core and NN-classifiers described above. The N-expert assumes that the target appears at most once in the picture. Hence, all positive patches that are too far from the selected bounding box are labeled as negative. Again, they can be used to update the core and NN-classifiers. The work [2] shows that even if the experts are not always correct, by using both of them we can balance the mistakes taken.

IV. Drone Control

Knowing the exact location of target in the captured picture, we can control the drone to keep the target always in the center of view. Note that we only know the relative location of target with respect to picture center and a scale with respect to the original bounding box. We will describe how to translate the scale to a distance to fly and then we describe the control procedure for the drone.

A. Scale Estimation

To control forward/backward movement we need to know the distance the drone is supposed to fly. Note that we know the scale of the bounding box with respect to the initial bounding box. Hence, if we know the size (for example height) of the target we can use trigonometry to compute the distance-to-go to keep the same distance from the target. However, this is not the case as the user can select any part of picture to be traced and we do not know its exact size (as it also depends on the distance and no stereo vision is used there).

Vice versa, if we know the distance travelled by the drone then we can compute the height of the target again using trigonometry (this simulates stereo vision by moving a single camera to a slightly different location). Figure 6 gives the idea of the method. The drone has some sensors, such as accelerometer and gyroscope, that can be used to trace the movement of the drone, at least in theory. There are methods how to do it with sensors available in the drone [5], but it is known that such measurements are not very accurate and deteriorate significantly with time (drifting). Unfortunately, even a small inaccuracy in the travelled distance measurement gives big discrepancy in the computed height so this approach is not appropriate as well.

At the end we decided to exploit simple reinforcement learning to learn what to do based on the scale of the bounding box. From the object tracking we obtain the size of the bounding box so we can compute a size relative to the initial bounding box. The goal is to keep the ratio equal one (same size). Obviously, when the ratio is greater that one, the drone should fly away from the target (backward); when the ratio is smaller than one the drone should fly towards the target (forward). Using reinforcement learning we learn the speed of flight for a given ratio where the reward is measured by the time to reach ratio one. For a given ratio we keep the speed learnt and next time, if we observe a similar ratio (difference is smaller than 0.08) then we reuse the stored speed of flight (action). Figure 7 shows how the error of expected distance is improving with the number of iterations.

B. PID Controller

To control the drone we use a standard PID controller [3]. The proportional-integral-derivative (PID) controller is a generic control loop feedback mechanism widely used in industrial control systems. Simply speaking, its goal is to dynamically minimize the error of the controlled system. The PID controller has three separate control parameters: the proportional, the integral and the derivative terms (Figure 8). The proportional, integral and derivative terms are functions of time and their interpretation is the following:

- The proportional term corresponds to the current error $e(t)$ of the system and is responsible for diminishing this error (difference between the setpoint and the current point). The proportional term is always required in a PID controller.
- The integral part represents the past errors $\int_0^t e(\tau)d\tau$ and is responsible for eliminating steady state errors. For a biased system a controller with no integral part may stabilize the system near the steady state, but never reach the steady state. One drawback of the integral term is the fact that it may slow down the process of reaching a setpoint.
- The derivative term is a prediction of the future error $\frac{d}{dt}e(t)$. Its key role is avoiding oscillations of the system around the steady state. Its magnitude is a function of the rate of change of the error in the system.

The weighted sum of these terms is then used to adjust the system actuators to get the right output values out at time t. The behavior of a PID controller is governed by the following equation:

$$out(t) = C_P e(t) + C_I \int_0^t e(\tau)d\tau + C_D \frac{d}{dt}e(t).$$

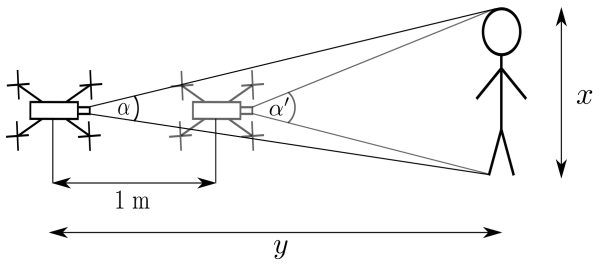

Fig. 6. Estimating the true size of the object of interest using simple trigonometry and the inertial measurement unit of the drone.

There are several techniques how to set the correct C_P, C_I and C_D constants. These constants set the magnitude of each of the terms in the PID controller and are specific for each application of the PID controller in a given environment.

C. Control Procedure

Recall that the drone can move left/right (by controlling roll), forward/backward (via pitch), up/down (change of altitude), and it can also rotate (by controlling yaw). Now, if the target moves right in the picture (which can be also caused by the drone flying left), there are two ways to follow the target: the drone can move right (roll) or the drone can rotate right (yaw) and then (if necessary) move forward, see Figure 9. We decided to implement the model where the drone uses rotation (yaw) rather than tilt (roll). The reason is that the drone does not "see" on its sides and there is a danger to hit some object staying next to the drone. Rotating the drone and then flying forward seems to be more safe as the drone sees the area where it is flying (if the target is visible, it is assumed that there is a free area between the drone and the target). For moving up and down, there is a similar danger, but we can only use altimeter (sonar) to deduce if the drone is too low and hence forbid descend. For climbing, we do not use any restriction.

First, we adjust the horizontal and vertical positions of the drone in such a way that the object of interest is seen in the center of the screen. This is easily achieved by computing the angle by which the drone needs to rotate around its vertical axis in order to get the object of interest in the vertical center of the screen. This angle is then given as input to the PID controller, that subsequently performs the right action. Similarly, we adjust the altitude of the drone so that the object of interest is seen in the horizontal center of the screen. If the drone is below the object, the controller instructs the drone to ascend approximately 0.4m. This process may be repeated several times, until the drone reaches the desired altitude. Similarly the controller instructs the drone when it needs to descend. Here we must be careful not to instruct the drone to descend when it is already flying too low as a contact with the ground might cause the drone to become uncontrollable. After we correct the horizontal and vertical displacement, we need to correct the possibly incorrect distance the drone has to the object of interest. This is done with the help of a scale estimator as described above. The scale estimator outputs a distance the drone shall cover in order to get in the right distance from the object of interest. This distance is given as input to the PID controller that then finishes the action. After each action we wait for a small amount of time (about a second) in order for the tracking algorithm to stabilize. The reason is, that while the drone is performing an action, the object tracking algorithm tends to output jittery bounding boxes.

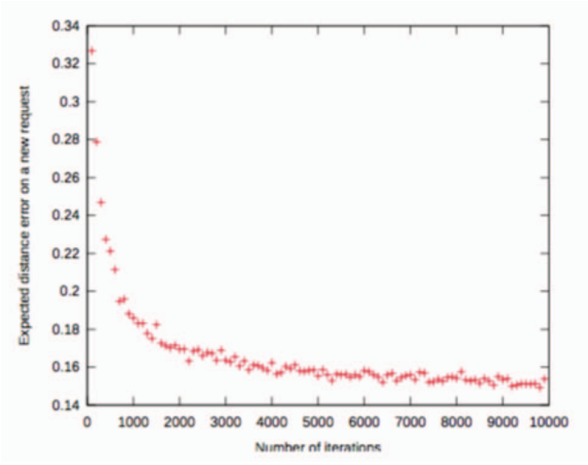

Fig. 7. Dependence of scale estimator error on the number of iterations. The scale estimator was learning actions for an object initially $3m$ away and the size of the object was $1m \times 1m$.

V. User Experience and Implementation

We have implemented the above-described techniques in the application *FollowMe* that is available for GNU/Linux and Windows operating systems. The application is written in C++ using various libraries to solve particular tasks. In particular we used Qt, SFML, OpenTLD, and Open CV. *Qt* (http://qt-project.org/) is a cross-platform application framework used for developing graphical user interfaces. We used the version Qt5. *SFML* (http://sfml-dev.org/) provides a simple interface to various components that a PC usually has. It is composed of five modules: system, window, graphics, audio and network. We exploited only the last module that provided us with simple send and receive functions over the UDP communication protocol. We used the version 2.1. *OpenTLD* (http://gnebehay.com/tld/) is an implementation of the TLD algorithm from [2]. As it lacks any API for external bindings we incorporated the source code of this software straight into our project and made some minor changes that would allow us to utilize this library right from our own code. Furthermore, we modified some constants that better suited the needs of our application. *OpenCV* (http://opencv.org/) is a cross-platform image manipulation library that is extensively used throughout the application, therefore it is necessary to have this library installed for the deployment of the FollowMe application. We used the version 2.4.

The goal of the *FollowMe* application is simplifying usage of the technology for the end user. The graphical user interface

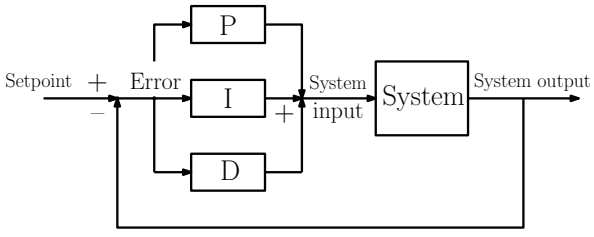

Fig. 8. PID controller block diagram.

(Figure 10) consists of a single window dominated by a panel showing the picture from drone's front camera (after connecting to the drone). In this panel the user can mark by a bounding box the object of interest that will be tracked by the drone. It is possible to initiate takeoff and landing there and the user can also manually control the drone via keyboard. The user can also see the trajectory travelled by the drone and some basic sensor information such as altitude, speed, and battery level.

VI. Conclusions

After years of separation, robotics and artificial intelligence are coming together and joining their effort in developing intelligent autonomous systems. Inexpensive robotics hardware and built-in sophisticated control mechanisms make it much easier for more AI researchers to implement and verify new techniques on real robots. One of the key words of this effort is integration. Many techniques have been developed separately and now it is the right time to integrate them to obtain more complex systems applicable in practice.

In this paper we showed how techniques originated in computer vision, machine learning, and control theory can be integrated to get software for autonomous tracking of any object selected by a user in a video frame and following that object by a flying drone. We used several existing techniques such as a PID controller, the Lucas-Kanade tracking algorithm, patch-variance, ensemble, and nearest neighbor classifiers, all integrated in the relatively novel tracking-learning-detection approach [2]. This integration is not only about putting the things together, but it requires careful tuning of parameters of integrated techniques and communication of right information at the right time. For example, blurred image going from drone's camera during movement causes failures of object deception algorithms. Hence it is important to make the drone steady for some time after performing the movement action so the target can be locked again before the next action. The presented idea was implemented in software *FollowMe* that makes it easy for end-users to use these sophisticated techniques on AR.Drone quadcopter. The goal of this paper

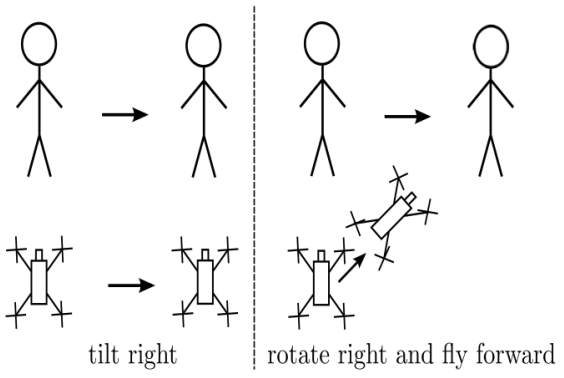

Fig. 9. Possible drone movements as a reaction to sidewise movement of target.

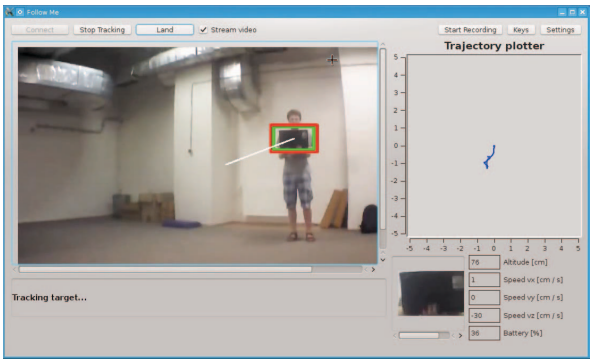

Fig. 10. Graphical user interface of the FollowMe application.

was discussing all aspects of developing such a software. Full technical details can be found in [8].

Acknowledgment

Research is supported by the Czech Science Foundation under the projects P103-15-19877S and P202/12/G061.

References

[1] Chi-Tinh Dang; Hoang-The Pham; Thanh-Binh Pham; Nguyen-Vu Truong: Vision based ground object tracking using AR.Drone quadrotor. In *Proceedings of 2013 International Conference on Control, Automation and Information Sciences (ICCAIS)*, pp. 146–151, Nha Trang: IEEE (2013).

[2] Kalal, Z.: *Tracking Learning Detection*. PhD thesis. University of Surrey, Faculty of Engineering and Physical Sciences, Centre for Vision, Speech and Signal Processing (2011)

[3] King, M.: *Process Control: A Practical Approach*. Chichester, UK: John Wiley & Sons Ltd. (2010)

[4] Lucas, B. D. and Kanade, T.: An iterative image registration technique with an application in stereo vision. In *Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, pp. 674–679, Vancouver (1981).

[5] Matzner, F.: *Tracking of 3D Movement*. Bachelor Thesis. Charles University in Prague, Faculty of Mathematics and Physics (2014).

[6] Pestana, J.; Sanchez-Lopez, J.L. ; Saripalli, S. ; Campoy, P.: Computer vision based general object following for GPS-denied multirotor unmanned vehicles. In *Proceedings of American Control Conference (ACC)*, pp. 1886 – 1891, Portland: IEEE (2013).

[7] Szeliski, R.: *Computer Vision: Algorithms and Applications*. Springer Verlag (2010) Available from: http://www.szeliski.org/Book/.

[8] Vyškovský. A.: *Object Tracking by a Flying Drone*. Bachelor Thesis. Charles University in Prague, Faculty of Mathematics and Physics (2014)