# End-to-End Visual Target Tracking in Multi-Robot Systems Based on Deep Convolutional Neural Network

Yawen Cui, Bo Zhang, Wenjing Yang, Zhiyuan Wang, Yin Li, Xiaodong Yi, Yuhua Tang
National University of Defense Technology, China
{cuiyawen16, zhangbo10, wenjing.yang, wzy, liyin13, yixiaodong, yhtang}@nudt.edu.cn

## Abstract

*The problem of one-on-one target tracking from a single monocular image acquired from the viewpoint of a follower robot itself is studied in this paper. Previous works mainly depended on locating, onboard sensors with control mechanism, while robot may not carry advanced onboard equipment for localization or GNSS may also fail in GNSS-denied/Indoor environments. In this paper we propose a novel approach based on a deep convolutional neural network called Deep-Track, which trains a supervised image classifier only using images captured by the camera in the follower robot. Specifically, the Deep-Track system can output the estimated velocity of the target as well as the velocity control for the follower, by operating merely on two adjacent frames. In order to verify the effectiveness of Deep-Track, we build up a large-scale dataset in the simulator, in which the performance of the Deep-Track is evaluated and it is shown that a high tracking accuracy is achieved.*

## 1. Introduction

In the research field of traditional computer vision, the problem of tracking is very common and many methods have been proposed for addressing the problem in certain scenarios, such as tracking a target from a video or in many images. Target detection, tracking and recognition are closely interrelated areas with significant overlaps. When tracking a target in a video, the target should be recognized and labeled in each frame of the video. Naiyan Wang et al. [1] cast this problem by training a stacked denoising auto-encoder offline to learn generic image features that are more robust against variations. Hyeonseob Nam et al. [2] propose a novel visual tracking algorithm based on the representations from a discriminatively trained Convolutional Neural Network. Tracking in many pictures also aims to identify the target. Fayao Liu et al. [3] propose to use the feature learning pipeline for visual tracking.

Tracking problem is also necessitated in a variety of multi-robot systems. For example, a police robot may track a criminal target or an unmanned aircraft may track an adversarial target [4], which may be formulated as one-on-one target tracking: a moving target is tracked by a moving follower. Solving the problem of one-on-one tracking is also important for many other applications, including flocking and formation control, where a robot may need to track and follow the leader robot [5].

Several previous works deal with target tracking in multi-robot system based on locating and control mechanism. Paolo Pirjanian et al. [6] propose an approach for multi-robot coordination in the context of cooperative target acquisition, which is based on multiple objective behavior coordination extended to multiple robots and may also be used in one-on-one target tracking. Some other works propose to use directional onboard sensors [7], [8] and implemented the directional sensing of electromagnetic waves [9].

For specific formation and flocking scenarios, some works also develop the tracking method by exploiting the prior knowledge exhibited in the target-follower geometrical relationships. For example, when a team of unmanned vehicles perch a line, a robot may identify the robot in front as the target and follow it. Kartik Mohta et al. [10] formulate the position-based visual servoing problem for a quadrotor equipped with a monocular camera and an IMU relying only on features on planes and lines in order to fly above and perch on arbitrarily oriented lines. However, in a lot of scenarios, unmanned vehicles may not carry advanced onboard equipment for localization, while GNSS may also fail in GNSS-denied/Indoor environments. Therefore, most methods proposed in the literature may fail or their performance may degenerate significantly.

Against the background, we propose a novel approach called Deep-Track in this paper that only depends on monocular images captured by a single forward-looking camera and cast the tracking problem in flocking and formation control as an image classification task. Specifically, we estimate the approximate velocity of the target only depending on the two images perceived by the followers in two adjacent frames. By adopting a supervised end-to-end machine learning approach based on Deep Convolutional Neural Networks (DCNNs), which operates directly on the image's raw pixel values, Deep-Track outputs action controls for the robot. Our main contributions of this paper are described as follows:

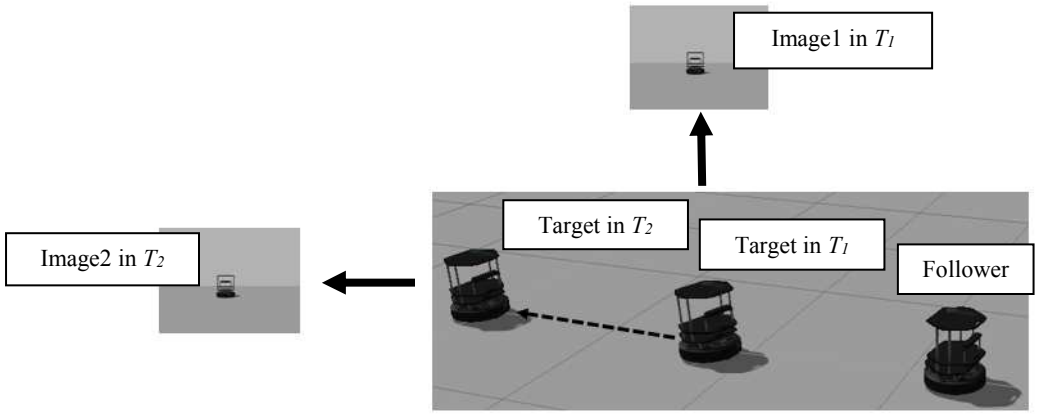



Figure 1: The target moves from the state of $T_1$ to the current state of $T_2$. Image1 and image2 are perceived by the follower in $T_1$ and $T_2$ respectively. After that, we construct a network to learn the difference and recognize the velocity of the target. Then, the velocity will be sent to the follower in order to keep tracking.

- Firstly, a one-to-one target tracking algorithm relying only on monocular images is proposed, called Deep-Track, is exempted from explicit determination on the characteristic features of the target.
- Secondly, a large-scale dataset is efficiently acquired in simulation environments, which is used for training and testing the Deep-Track algorithm.
- Finally, remarkable tracking performance is achieved by the proposed Deep-Track.

## 2. Visual tracking of the target

In order to successfully track a target, a robot has to perceive where the target is and how it moves from the last time step, then react in order to maintain the distance from the target. In this paper, we propose a DCNN-based approach for one-on-one visual target tracking and show experimental results on an autonomous robot. We consider two monocular images in two adjacent frames from a forward-looking camera as inputs, as illustrated in Figure 1. Using a single monocular image may also train a classifier with lower computational complexity, however the decision is not reliable as it exploits very little 3-D information for capturing the velocity. Our method of using two adjacent images may be seen as the simplest form for image-sequence based classifier, which extracts 3-D information [11] by exploiting the visual difference in adjacent frames, achieving higher classification reliability.

When a target moves with a certain velocity, there are rules to follow which is about the differences of some pixels between images perceived by the followers in two adjacent frames (see in Figure 2). We may adopt machine learning to acquire these rules and cast the tracking problem as an image classification task by labelling input images with different velocities.

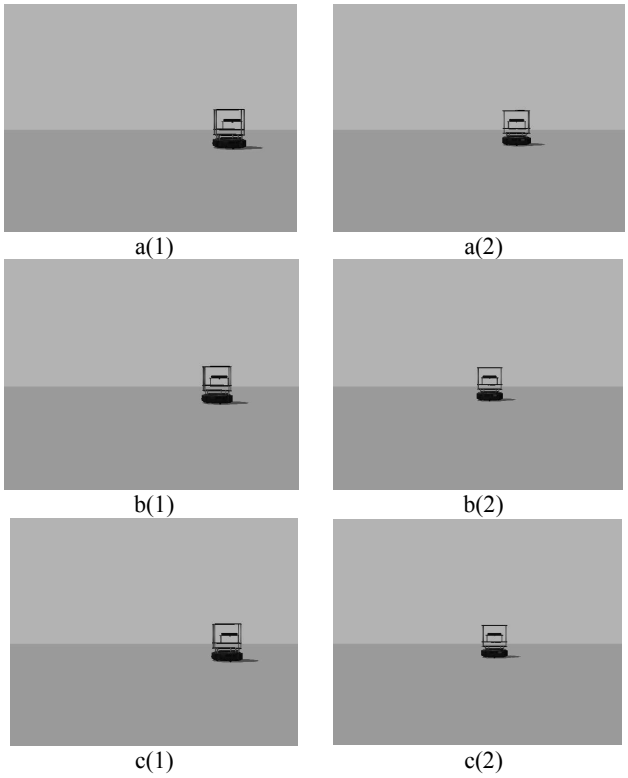



Figure 2: The target moves from the state of last time step T1 to the state of current time step T2 with the linear velocity of 0.2 m/s and the angular velocity of 0.2 m/s. There are three pairs of data. Image1 (the left one) and image2 (the right one) are perceived by the follower Turtlebot in T1 and T2 respectively.

As for this tracking problem, it may also be suitable to use regression because the velocity of robot is continuous. However, this paper considers a demo with affordable computational complexity, by casting control problems of robots as classification tasks as in [12] and [13]. In our future work, a comprehensive comparison between regression and classification based approaches will be carried out.

During our experiment, we choose Turtlebot-2 as the target and the follower. TurtleBot-2 is a low-cost, personal robot kit with open-source software. It was created at Willow Garage by Melonee Wise and Tully Foote in November 2010. With TurtleBot, we are able to build a robot that can drive around the indoor environment, possess enough horsepower to create exciting applications. More importantly, its full-3D model is available in simulators, which captures the sensing, control, kinematics and dynamics behavior of TurtleBots.

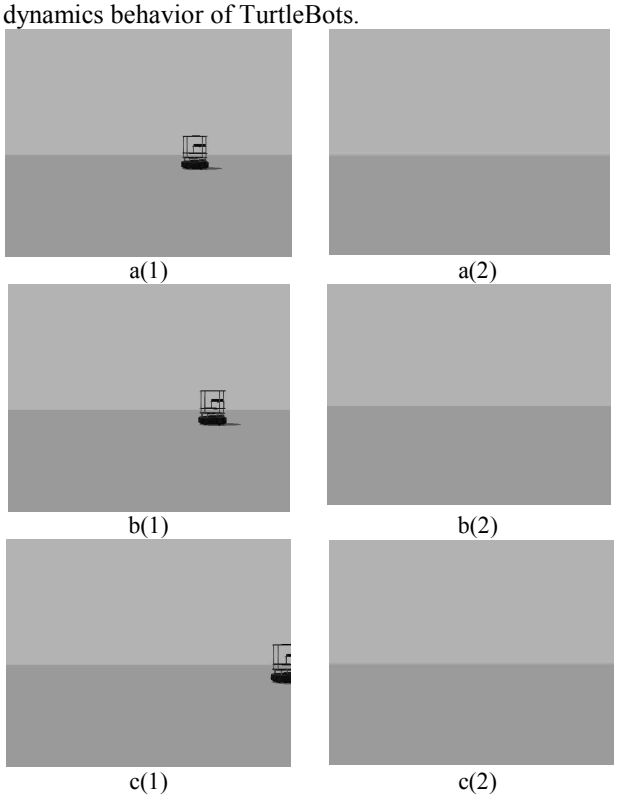



Figure 3: There are three pairs of images that are collected when the linear velocity is 0.5 m/s and the angular velocity is 0.5 m/s. When the target robot moves, it may get out of the view of the camera on the follower robot easily (See the second image of each pair of data).

Due to the limitation of robotic maneuverability and in order to track accurately, we consider a discrete set of velocities for the target robot: the range of the linear velocity and the angular velocity are (0.0 m/s, 0.5 m/s] and [-0.3 m/s, 0.3 m/s] respectively. The two kinds of velocities are not within the same range because target tracking may fail when the angular velocity of target is too large and beyond the perceiving capabilities of the camera on the follower robot (See in Figure 3). The follower robot can change its pose until the camera is able to perceive the target robot, but this process is time-consuming, thus cannot meet the real- time requirement. Therefore, we consider a velocity set that contains eight velocities for the sake of classification, as illustrated in Figure 4.

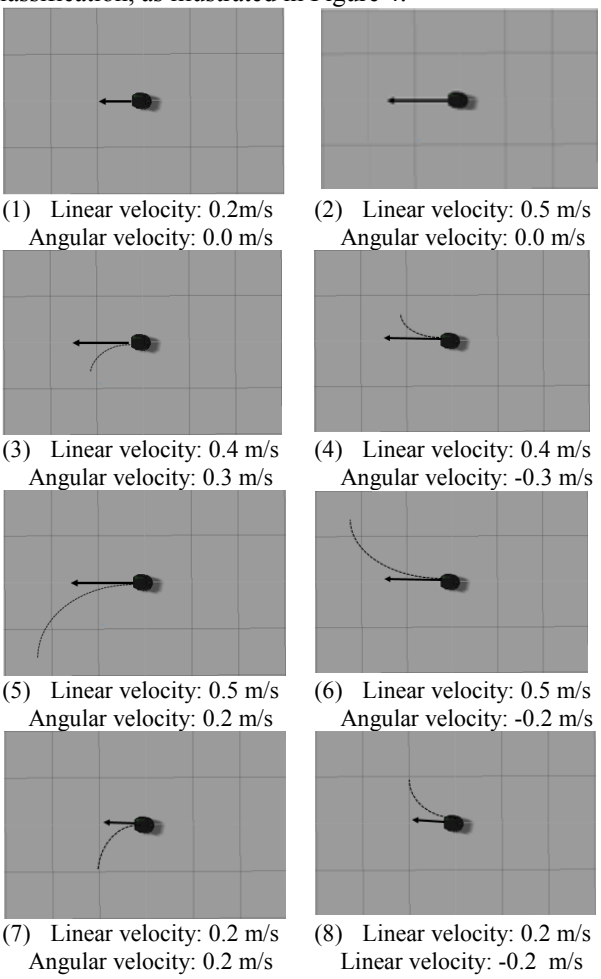



Figure 4: Illustrations of the selected velocity classes.

## 2.1. Dataset

One of the advantages of Deep-Track in one-on-one target tracking is that the information about the distances and positions of robots can be effectively and directly extracted from the visual images. The DCNN-based classifier in Deep-Track needs to be trained to achieve satisfactory performance and there is no dataset readily available, hence a wide range of data is collected with different poses, positions and distances between the target and the follower.

Each pair of images is classified according to its Euclidean distance from the candidates in the velocity

classes, i.e. associated to its ground-truth class. All images are perceived by a forward-looking camera and we collect the dataset in the scenario illustrated in Figure 5. The dataset is composed by 8000 pairs, namely, 16000 images collected at 20 positions. For a certain pair of the images generated in the dataset, the follower robot captures the 1-st image of the target, then a sequence of velocities is sent to the target robot in the next time step and the target robot moves. At the meantime, the follower robot captures the 2-nd image of the target, then adopts the same velocities as the target. In this paper, we use a vector to represent the velocity: [linear velocity, angular velocity].

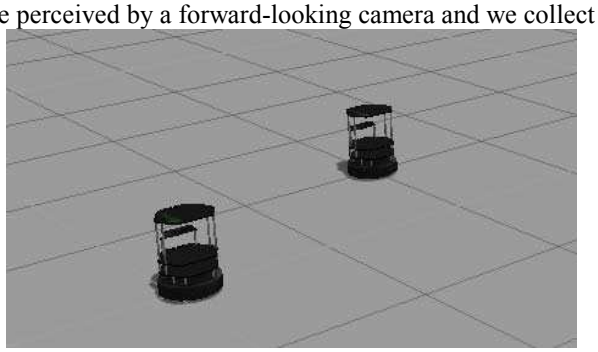
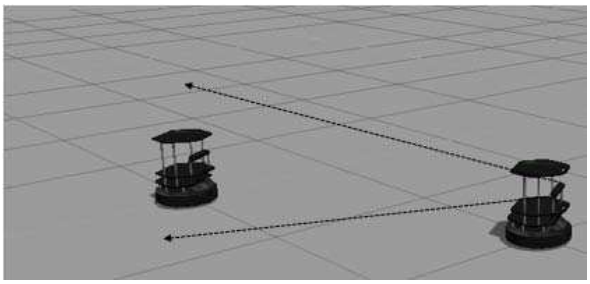
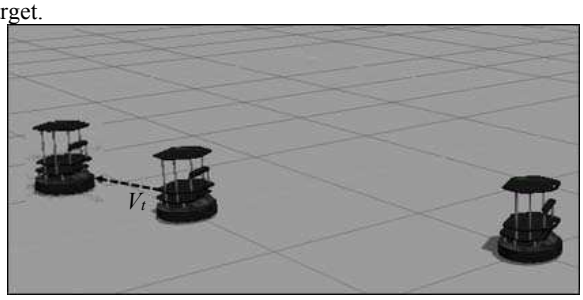


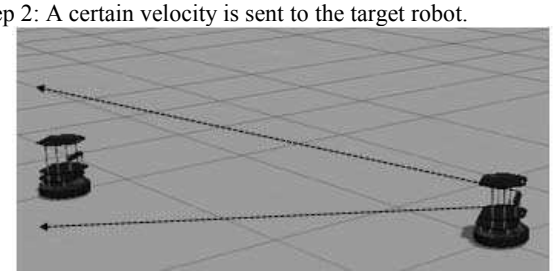
Figure 5: This is the scenario when we collect the dataset and there is no obstacle.



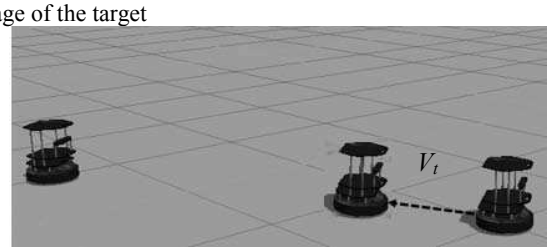
Step 1: At first, the follower robot captures the 1-st image of the target.




Step 2: A certain velocity is sent to the target robot.



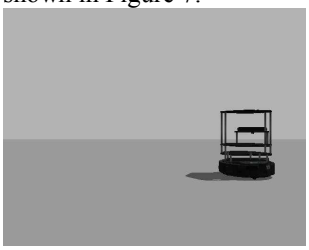
Step 3: At the meantime, the follower robot captures the 2-nd image of the target



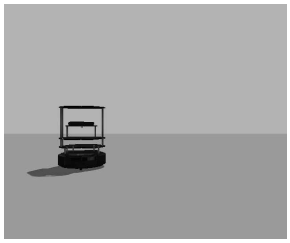

Step 4：The follower robot adopts the same velocities as the target.

Figure 6: The process when we collect the dataset.

A labeled data is composed of two images perceived by camera on the follower, along with the label indicating its ground-truth velocity within the velocity class. The process is described in Figure 6 and three examples of data are shown in Figure 7.

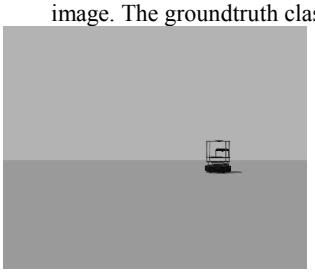


a.(1)

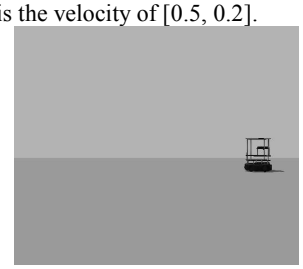


a.(2)

(a) Left image is the first one and right image is the second image. The groundtruth class is the velocity of [0.5, 0.2].

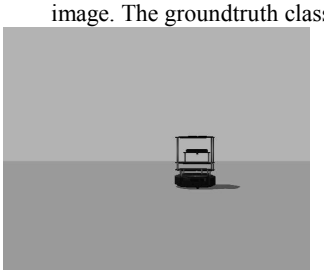


b.(1)

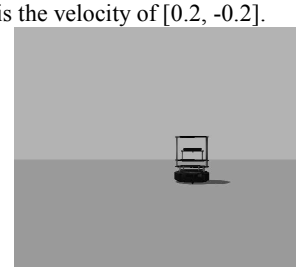


b.(2)

(b) Left image is the first one and right image is the second image. The groundtruth class is the velocity of [0.2, -0.2].



c.(1)



c.(2)

(c) Left image is the first one and right image is the second image. The groundtruth class is the velocity of [0.5, 0.0].

Figure 7: two examples of data.

## 2.2. Deep Convolutional Neural Network for Tracking

Based on the labeled dataset, we address the tracking problem as a supervised machine learning task. We use DCNN as the image classifier, and design its architecture as shown in Figure 8. We consider two matrices with a size of


Figure 8: The architecture for the Deep Convolutional Neural Network used in our method. The input image $I_1$, as well as image $I_2$, are all fed into a 6 × 6 convolution with stride 2, followed by a 5 × 5 convolution and a 3 × 3 max-pooling. This is followed by a 5 × 5 convolution and a 3 × 3 max-pooling again. Then they are merged to the size of 27 × 27 ×64 using concatenation. The result is then processed by three 5 × 5 convolutions, two 3 × 3 max-pooling layers, and four fully connected layers after which the network outputs the classification result.

3×472×472 as inputs, followed by several hidden layers and eight output neurons. The input image pair is firstly resized from 640×480 to 472×472 pixels, and the resulting 3×472×472 RGB values are directly mapped to the neurons in the input layer.

Then, the images pass through a two-stage DCNN, where the first stage includes two independent channels, while the second stage merges the two channel outputs using the operation of concatenation and extracts the difference feature together with the output of the classification results. As a first remark on the DCNN design, we divide the DCNN into two channels at first. As for two channels, we extract features of the first image and the second image respectively. It can be more efficient and targeted when we extract features than merging the two images at first.

The two stages are described as follows:

● First-stage: Each pair of images is passed through the network respectively to extract the features of each image. After 6 × 6 convolution with stride 2, followed by two 5 × 5 convolutions and two 3 × 3 max-pooling layers, they are compressed to feature vectors having the size of 27 × 27 ×64.

● Second-stage: In order to extract the features of differences between two images in one pair, the result after concatenation is then processed by three 5 × 5 convolutions, two 3 × 3 max-pooling layers, and three fully connected layers with 8 units, after which the network outputs the classification result. For a given input, the DCNN outputs eight values, representing the probability that the input has for each velocity class.

The network takes the image $I_2$ of the current time step $T_2$, together with an additional image $I_1$ perceived in the last time step $T_1$ as inputs. This additional image is used to be compared with the current image and they are all passed through the hierarchical classifier and the classifier outputs the velocity of the target according to raw pixels. In order to keep tracking, the velocity retrieved from the network is then sent to the follower.

Implementation details of the layers in the Deep-Track DCNN is given below:

1. Firstly, layers of the first-stage are all processed by the batch normalization except concatenation layer. We use a decreasing learning rate from 0.01 to 0.0001 and the iterations of training phase is 30000.

2. Secondly, the convolutional layer performs 2D convolutions of their input channels with a rectangular filter [14]. If there are several channels in the previous layer, the

results of the corresponding convolutions are summed and 3. the filter better matches the content of the map, a higher activation is given.

4. Thirdly, the max-pooling layers [15] also referred to subsampling layer are adopted for decreasing the map size, thus reducing the network complexity.

Finally, the output layer is a fully connected layer with one neuron per class activated by a soft-max function. Each output neuron's activation can be interpreted as the probability of the input image belonging to that class. We train this network with a Cross-Entropy loss and using Gradient Decent method for optimization.

## 3. Experimental results

In order to validate the effectiveness of our method, we first choose an environment to collect the data, then construct, train and test the DCNN in Deep-Track.

### 3.1. Experimental setup

As for the first work, we form our dataset in the simulator named Gazebo [16] based on Robot Operating System (ROS) [17]. The ROS is a set of software libraries and tools that enable us to build robot applications. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environment. In ROS, programs exist in the form of nodes and message exchange occurs among nodes when one node publish or subscribe topics. Considering formation or flocking applications, we assume that the target and the follower are at the same pose all the time.

The developed message publishing and subscribing passed by a scaled hyperbolic tangent activation function. If structures in data collection process and testing process in ROS are illustrated in Figure 9 and Figure 10 respectively.

(1) The data collection process:

As shown in Figure 9, there are two modules, the data collection module and the simulation module. In the data collection module, node *A* is responsible for collecting the image data consisting of multiple image pairs denoted by $I_1$ and $I_2$, while node *B* is responsible for collecting the image labels, namely the velocities of the target robot. In the simulation module, node *C* and node *D* simulate the behavior of the follower and the target robot in the Gazebo environment respectively.

The detailed steps of the data collection process are given below:

- Firstly, node *A* subscribes the topic referring to message of the camera in the follower robot and stores the image as $I_1$.
- Then, node *E* publishes the topic referring to the velocity of the target for 5-second, while the target robot implemented by node *D* subscribes the topics.
- Afterwards, node *A* subscribes the topic referring to message of the camera in the follower robot again and stores the image as $I_2$.
- The topic lasting 5-second about the velocity of the target/follower is published by node *E* again, while the follower robot implemented by node *C* subscribes the topic in order to keep track of the target robot.

Following the process indicated in Figure 10, we collect 8000 pairs, 16000 images in one scenario that are described in Section 2.

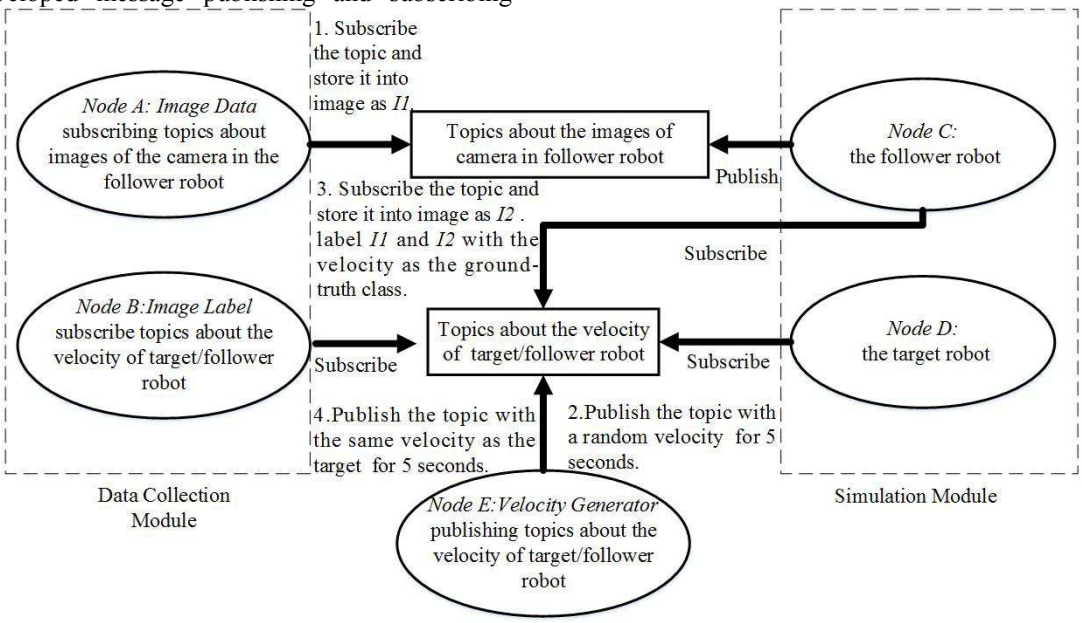



Figure 9. The nodes relationship graph for the training process.

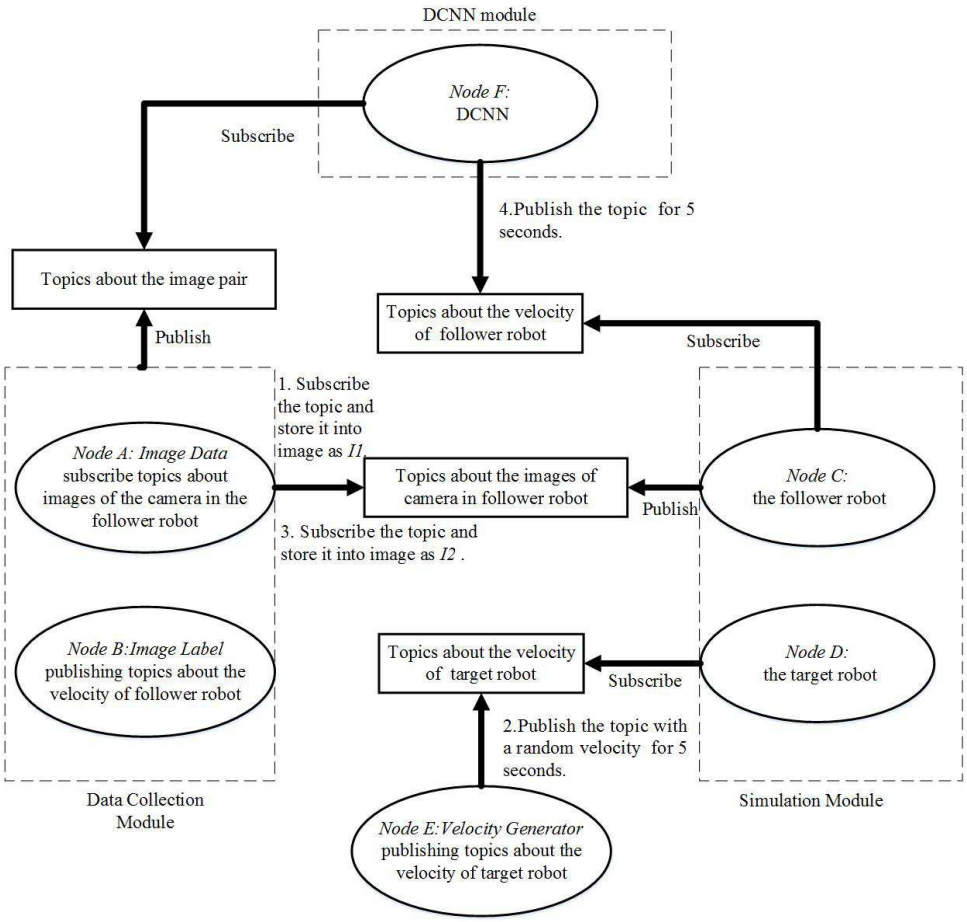



Figure 10. The nodes relationship graph for the testing process.

(2) Testing process

The nodes relationship graph for the testing process is shown in Figure 9. Compared to the data collection process, the major difference implemented in the testing process is that an additional module DCNN is adopted, which is the core module in Deep-Track.

The DCNN subscribes the images captured by the camera of the follower robot, classifies its class and hence gets the velocity of the follower robot. In comparison, in the data collection process, the velocity of follower robot is directly generated from node E.

When constructing the model of DCNN, we program it with the library of TensorFlow [18] which is an open source software library for numerical computation using data flow graphs. Computation nodes in the data flow graph represent mathematical operations, while the graph edges represent the multi-dimensional data arrays (tensors) that flow between them. When we construct the model of DCNN, roughly five steps are required:

- Define the layers: convolutional layer, pooling layer and normalization layer.
- Define the variables of weights and bias and construct the graph using the layers that are defined before. The images used as input and labels are all described as placeholders.
- Define loss, accuracy and optimization function.
- Initialize all variables. Create the session to run the graph and feed data to placeholders.
- Create circles to optimize the loss using the method of Gradient Decent.

In the period of training, A GPU with the memory of 6G is used to speed up training.

### 3.2. Results

During the training stage, our experiment involves 5-fold cross validation. Specifically, the dataset is divided into five folds of equal size, according to the positions of two robots with same poses. The DCNN model is trained based on four of the folds and tested in the remaining fold. We repeated the process five times, for each an individual fold as test data and computed the average accuracy of each model in three iterations as its final result. The result is shown is Table 1

| Iteration | 1 | 2 | 3 | 4 | 5 | Average |
|---|---|---|---|---|---|---|
| Accuracy | 0.783 | 0.813 | 0.750 | 0.807 | 0.778 | 0.786 |

Table 1: The classification accuracy after training.

### 3.3. Discussions and future work

We take a further analysis by looking into the cases of failure. Figure 11 reports three of the most representative failure cases.

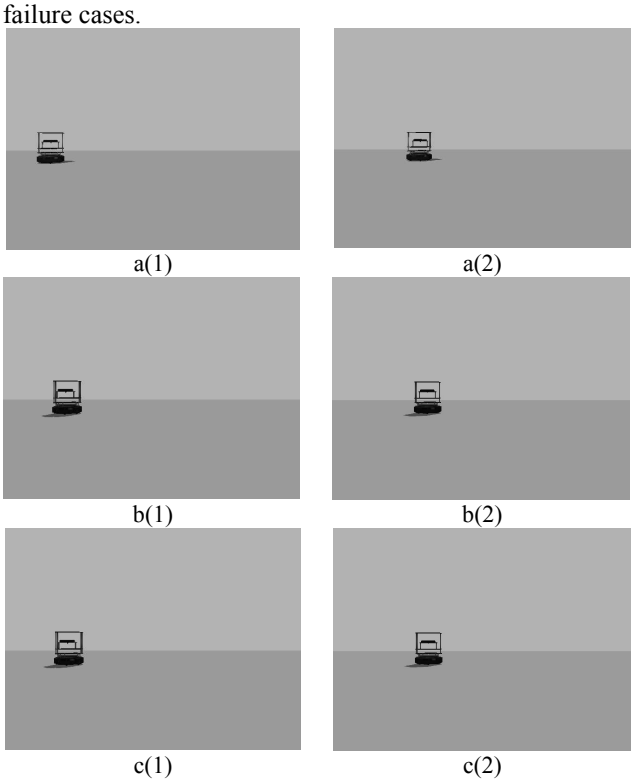



Figure 11: Two failure cases, where the ground-truth of the velocity is [0.2, 0.2] and the network all outputs the class of [0.4, 0.3].

Hence, our future work will be devoted to addressing the following limitations of the Deep-Track:

(1) The set of velocities is limited to eight discrete classes, which should be extended for practical fine-grained tracking.
(2) The environment around Turtlebots is relatively simple. Further work will incorporate more sophisticated environment models.

With regard to the limitations, we will do the next work:

(1) We will collect the dataset in a larger scale of positions and velocities.
(2) More kinds of collisions will be put into the environment including moving objects.
(3) We will find a mechanism for continuous servoing when the robot tracks the target that moves continuously.

## 4. Conclusions

In this paper, we propose an end-to-end visual target tracking method called Deep-Track in Multi-Robot Systems using deep convolutional neural networks (DCNNs). Deep-Track only relies on monocular images captured by a forward-looking camera and cast the tracking problem as an image classification task. By operating the raw pixels directly, we train a DCNN to complete the classification task with a large scale dataset. The performance of the Deep-Track is evaluated and it is shown that a high tracking accuracy is achieved.

## 5. Acknowledgement

This work was supported by the National Natural Science Foundation of China under Grant Nos. 91648204, 61532007 and 61601486, the Research Programs of NUDT under Grant No. ZDYYJCYJ20140601.

## References

[1] N. Wang D. Yeung, “Learning a Deep Compact Image Representation for Visual Track,” Advances in Neural Information Processing Systems, Stateline, Nevada, 2013.

[2] H. Nam, B. Han, “Learning Multi-Domain Convolutional Neural Networks for Visual Tracking,” Conference on Computer Vision and Pattern Recognition, Las Vegas, Nevada, 2016.

[3] F. Liu, C. Shen, I. Reid, A. Hengel, “Online Unsupervised Feature Learning for Visual Tracking, ” Image and Vision Computing, vol.51, pp. 84-94, Jul. 2016.

[4] J. S. McGrew, J. P. How, B Williams, N. Roy, “Air-Combat Strategy Using Approximate Dynamic Programming,” Journal of Guidance Control and Dynamics, Vol. 33, No. 5, Sep.–Oct. 2010.

[5] Z. Cai, X. Chang, Y. Wang, X. Yi, X. Yang, “Distributed control for flocking and group maneuvering of nonholonomic agents,” Computer animation & virtual worlds, vol.28, Issue 3-4, May/Aug. 2017.

[6] P. Pirjanian and M. Mataric, "Multi-robot target acquisition using multiple objective behavior coordination," International Conference on Robotics and Automation, San Francisco, CA, 2000.

[7] M. Mazo, A. Speranzon, K. H. Johansson and Xiaoming Hu, "Multi-robot tracking of a moving object using directional sensors," IEEE International Conference on Robotics and Automation, New Orleans, LA, 2004.

[8] Karol Hausman, J Müller, Abishek Hariharan, Nora Ayanian and Gaurav S Sukhatme, “Cooperative multi-robot control for target tracking with onboard sensing,” The International Journal of Robotics Research, Vol. 34(13), pp. 1660–1677, 2015.

[9] G. Lee, K. Tatara and N. Y. Chong, "Hardware-assisted direction estimation for mobile robot target tracking applications," 2015 IEEE International Conference on Mechatronics (ICM), Nagoya, 2015.

[10] K. Mohta, V. Kumar and K. Daniilidis, "Vision-based control of a quadrotor for perching on lines,"2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, 2014.
[11] David A. Forsyth and J. Ponce. Computer Vision: A modern Approach. Prentice Hall, 2003.
[12] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, D. Quillen, "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection," The International Journal of Robotics Research, Jun.2017.
[13] A. Giusti, J. Guzzi, D. Cireşan, F. He, J. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, et al., "A machine learning approach to visual perception of forest trails for mobile robots," IEEE Robotics and Automation Letters, vol. 1, no. 2, pp. 661–667, 2016.
[14] Y. LeCun, L. Bottou, G. Orr, and K. Muller, "Efficient backProp," New York, NY, USA: Springer, 1998, pp.9-50.
[15] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," International Conference on Artificial Neural Networks, Thessaloniki, Greece, 2010.
[16] D. Roberts, R. Wolff and O. Otto, "Constructing a Gazebo: Supporting Teamwork in a Tightly Coupled, Distributed Task in Virtual Reality," Presence, Volume: 12, Issue: 6, Dec. 2003.
[17] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R.Wheeler, and A.Y. Ng, "ROS: An open-source robot operating system," Proc. ICRA Open-Source Softw. Workshop, 2009.
[18] M. Abadi, A. Agarwal et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," arXiv:1603.04467, 2016.