Real-Time Action Recognition System from a First-Person View Video Stream

Maxim Maximov, Sang-Rok Oh, and Myong Soo Park

***Abstract*—This paper presents an approach to recognizing people's actions directed towards a robot, from a video collected by a camera positioned on the robot. Our approach has two modes: "stand by" mode – when a system waits for an action to start and "active" mode – when the system recognizes actions from the video stream. "Stand by" mode allows to save resources during an absence of actions and also gives a starting point of an action, which helps for recognition. Our action recognition model is based on a Bag-of-Words model, but we utilized sparse features in order to process video frames faster. As a result, the system implemented with this approach could continuously recognize actions in real-time using fewer computations. The performance of our method was experimentally evaluated and the results are given at the end of the paper.**

***Index Terms*—Human action recognition, real-time video processing, bag-of-words, computer vision.**

I. Introduction

The goal of this paper is to present a method for an action recognition using the first-person view (FPV) videos. The FPV video is a video made from a point of view of an actor interacting with surrounding environment. The actor can be a human, an animal or a robot. The FPV videos can be divided into observational and egocentric types: as the name states, the observer FPV video has an actor that observes an action that directed towards it, while the egocentric videos contain actions made by an actor. From this point, we refer the observer FPV as the FPV and the egocentric FPV as the egocentric view, and our work focus on the observer FPV.

The recent popularity of wearable cameras and the progress in robotics have created an interest in the action recognition from the FPV. One of the prospective application fields of the FPV action recognition is the human-robot interaction (HRI). A robot needs to understand human behavior, particularly, the actions towards a robot, to make an appropriate response. These actions include not only command gestures and some usual actions such as waving, but also the harmful actions towards a robot. To achieve such result, the FPV action recognition can play an important role.

When we use the FPV-based action recognition in HRI, there are some requirements to be considered. Many algorithms focus on recognition of finished actions from videos with limited numbers of frames, while for HRI, we need a way to recognize actions in real-time from a continuous video stream. Also, in many situations, an action recognition task is not a priority for a robot, so computational power designated to this task might be very limited.

In order to meet the above-mentioned requirements, we designed a new system for FPV-based action recognition. The main part of this system is an action recognition model and based on a Bag-of-Words (BoW) model [1], also called as a Bag of Visual Words (BoVW), which is one of the widely-used models in the action recognition field.

Instead of using the BoW model as it is, we sought to find a model with an improved configuration for our purpose to work in the real-time even with limited computational resources. We achieved this task by empirically testing changes in performance by modifying different components of a BoW model. Also, we added a subsidiary model that recognizes a start and an end of an interaction between camera holder and a person. This subsidiary part goals are to lessen computations when no interaction is happening and to improve an action recognition performance by giving a starting point of an interaction.

The remainder of this paper is organized as follows. Section II describes the background of this paper, the related works and the system overview, and Sections III and IV present two core components of our system (for pre-action and the action) with the configuration of other components. Evaluation and comparative study are conducted with the setup described in Section V and the results are given with discussions. Then, Section VI concludes this paper.

II. Background

In this section, we will provide background knowledge to be helpful in understanding of the subsequent sections: related works and system overview.

A. Related Work

A lot of works have been done in the action recognition area. Many of them focused on recognition from surveillance camera videos, movies and internet video clips, and only a few focused on the FPV. One of the first researches on the action recognition towards camera presented in the [2] paper. Actions of humans were directed towards a camera, which was mounted on a robot. Their system adopted the BoW model, and used two different types of features. The performance of this system was evaluated for 7 actions using their dataset that was publicly available and also used for our system. However, their system has limitations: long computation time due to using dense and high dimensions'

Manuscript received August 10, 2015; revised January 10, 2016.

M. Maximov is with the Interaction & Robotics Research Center, Korea Institute of Science and Technology, Seoul, Republic of Korea and also with Korea University of Science and Technology, Republic of Korea (e-mail: maxsqr@gmail.com).

S. R. Oh is with Korea Institute of Science and Technology, Seoul, Republic of Korea (e-mail: sroh@kist.re.kr).

M. S. Park is with the Interaction & Robotics Research Center, Korea Institute of Science and Technology, Seoul, Republic of Korea (e-mail: meister1@gmail.com).

DOI: 10.7763/IJCTE.2017.V9.1116

features, and applicability to premade videos clips, not to video stream.

Ref. [3] used attention as a feature to recognize human social interaction from FPV. The useful aspect is their attention detection method, that detects faces and their angles to determine a direction of their gazes. Observing people attention direction can be later used in the FPV to recognize a start of an interaction: if a person looked at a camera direction, then there is a high probability of an interaction.

Human-robot interaction work [4] often use approaches based on detecting and tracking human face/limbs using depth information. Although, depth information can be quite useful, methods based on human limbs and pose detection are not reliable due to an often case of partial visibility of human body. In FPV, people often come very close to a robot and do not fit in their field of view.

Action recognition in movies scenes and various videos from mobile cameras became popular recent years. The popularity of YouTube kind of web services raised the number of publicly available mobile videos to an enormous amount. And many movies scenes and various video clips can also be considered as the FPV videos, because they often recorded from 3rd person observer position, or sometimes, whole or part of the movie shot in a first-person perspective. So we can adopt successful action recognition methods used in this kind of works [5], such as BoW model, optical flow and spatiotemporal point of interests.

The BoW model is widely used in the action recognition field [2], [5]-[7]. [2] improved BoW model and applied it for the FPV videos. The [5] showed a possibility of action recognition in movies using BoW model from movies and using sparse-temporal features [8]. Recently, [6] have done extensive research on the BoW model for the action recognition.

B. System Overview

Unlike previous works, we did not continuously recognize actions in front of a camera. To resolve processing continuous videos problem, we segment an action into two successive parts: pre-action and action. Each part is different from others and has its purpose. The pre-action part contains what happens before actions occur, that is, when people do not interact with a robot - the "stand by" mode. Once the system recognizes the possibility of interaction with a human, it decides that the action part has started. The action part consists of a person action in front a robot and an effect of this action on the robot. Effects of the harmful actions are usually more distinguishable because they cause turmoil in a camera view. We considered these two parts of an action in our system. The system has two corresponding modes: "stand by" and "active". In our system, we created the "pre-action" subsystem that tracks changes in the environment from the camera and decide whether we should start to recognize an action - switch to "active" mode, or finish to recognize - switch to "stand by" mode.

Our system is composed of two sub-systems for dealing with pre-actions and actions, respectively. The proposed pipeline of the system between these sub-systems is presented in Fig. 1. Note that the action recognition part treats local and global features separately from each other until classification step, details and meaning of which will be described in the next section.

In this paper, we assume that videos are taken from a camera that is mounted on a mobile robot, which implies smooth camera motions and frontal field of view of a robot. Also, during an action towards the robot, the robot is assumed to stay still and fully observe the action in front of it. There is no other special assumption.

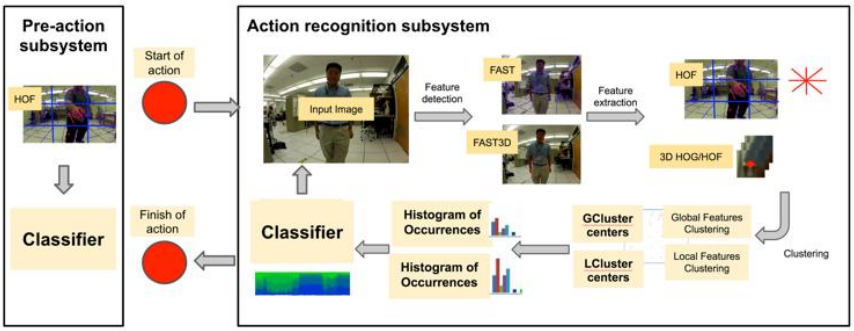

Fig. 1. The system pipeline.

III. Pre-action Subsystem

In this Section, we present pre-action subsystem. To work continuously and use fewer resources, we integrated the system mode that works when a robot doesn't interact with people. It is the "stand by" mode when robot only wait for an interaction to start and at the time an interaction with a robot is about to start, the "pre-action" subsystem switch to the "active" mode and vice versa.

Using the pre-action step, our system aim is to automatically detect a starting point for transition to the action recognition "active" mode and ending point to switch to "stand by" mode.

To detect start and finish of actions, we decided to use features from a frame that can give us information about a movement of people in front of a camera. We assume that if a person is coming closer to a camera, then he is going to start an interaction with it. We present two kinds of features that give us such information: Optical Flow and Bounding Box features.

A. Feature Extraction

1) Optical flow

Optical Flow (OF) is a widely-used method that represents a motion in a video frames sequence. The OF methods track directions and magnitudes of offsets of given points between two consecutive video frames. In our system, a big magnitude of an OF indicates that a person is close to a robot and going

to interact with it, and a small magnitude indicates that a person is far from a robot.

There are two main types of OF: dense and sparse. The dense OF methods track offsets of a grid of pixels between two video frames, and the sparse OF methods track offsets only from a set of detected input points. The set of input points can be generated by utilizing interest points detectors such as corner detection algorithms. Most of the dense OF methods show a better precision, but a greater computation time, which makes them less suitable for the real-time purposes, especially with limited computational resources. Thus, we apply a sparse OF method in our system.

Every several number of frames, the system detects $P_{det} = [p_1, \ldots, p_d]$, a set of interest points for a sparse optical flow using the FAST corner detection algorithm [9], where d is a number of detected points and $p_i \in R^2$ is a position of detected point i. We used the FAST detector, because it shows one of the fastest computation time among interest points detection algorithms, along with a distinctiveness of detected points [10].

After points detection, in the next consecutive frames, the system tracks offset of detected points from P_{det} using the Lucas-Kanade method. From resulted offsets in each frame, our system extracts angles of offsets directions τ_i and offsets magnitudes mag_i for $i \in [1..d]$. Fig. 2 shows an example of the sparse OF.

Raw OF consists of a lot of offsets vectors extracted from each frame, and separately, they give almost no useful information about whole person motion in a frame. Therefore, we summarize all OF vectors in a frame in one feature vector by using a Histogram of Optical Flow (HOF) method. A HOF is an OF descriptor that is commonly used in the action recognition. Firstly, all OF offsets are clustered to one of the bins h_j for $j \in [1..N_{bin}]$ depending on their τ_i, where N_{bin} is a number of bins. Each bin represents one general direction of offsets inside a bin. Next, a HOF is presented as a histogram $H = [h_1, \ldots, h_{N_{bin}}]$, where each bin value is $h_j = \sum_{k=1}^{K_j} mag_k$ and K_j is a number of OF offsets belonging to a bin j. Finally, because Sparse OF has an uneven distribution of detected points in each bin, we normalize a sparse HOF by dividing each computed bin to a number of offsets in this bin: $norm(h_j) = {h_j}/{K_j}$ for $j \in [1..N_{bin}]$, since the sparse OF has an uneven distribution of detected points.

Fig. 2. Example of Sparse OF (Blue dots represent detected points, and red lines represent offsets of these points).

To have more detailed information, we divide each video frame to $n \times m$ blocks. For each block, the system computes a HOF for a N_{bin} number of directions. We use resulted HOF vectors as an input to a classification model.

2) Bounding box

In this approach, the system detects people in a single video frame using the Histograms of Oriented Gradients (HOG) [11] method. Then, it constructs bounding boxes around detected people and extracts features from them. This approach heavily depends on a performance of the human detection algorithm. Also, besides features extraction, a bounding box can also be used for other tasks, for example, to extract foreground around detected humans or for human recognition.

Extracted features are given for each bounding box b_i as follows: a width $b_{i,w}$, a height $b_{i,h}$, and a deviation of b_i center from the center of a frame $b_{i,d}$. These are simple features that give us information about a size and position of a bounding box. Bigger bounding boxes of detected people are assumed to indicate that these people are closer to a camera. Also, wider boxes can mean that these people are facing a camera and making an action with stretched arms. And as we mentioned earlier, we assume that a person will stand in front of a camera (close to the center of a frame). Therefore, a deviation from the center value can tell how far is the person from the camera.

To be able to work with different frames resolutions, the system extracts previously mentioned features as a proportion to their maximum possible values: $\frac{b_{i,w}}{f_w}$, $\frac{b_{i,h}}{f_h}$, and $\frac{b_{i,d}}{\sqrt{(f_h/2)^2+(f_w/2)^2}}$, where f_w and f_h means a width and a height of a frame.

Resulted features values are concatenated to one feature vector $F = [\frac{b_{i,w}}{f_w}, \frac{b_{i,h}}{f_h}, \frac{b_{i,d}}{\sqrt{(f_h/2)^2+(f_w/2)^2}}]$ and then, F is given to a classification model input.

Additionally, we tested a face detection method using Haar features instead of a person detection. The face detection has an advantage in an attention detection: we can't detect a frontal face if a person turned away from a camera, which means that the person is not interacting with the camera. Unfortunately, the face detection methods are unstable in the low-resolution videos and in situations when a person located in a far distance.

B. Classification

In the classification step, we had training and testing phases, each with its dataset. A dataset is a set of features vectors extracted from the previous step and corresponding class value for each feature vector.

For a classification, we used a binary, supervised Random Forest (RF) model to classify video frames between 2 classes: "pre-action" and "action". Random Forest is a learning method that uses an ensemble of decision trees. Each decision tree trains on a random subset of a whole training dataset and gives a single classification result for each feature vector of a testing dataset. RF decide final classification result, for each feature vector, by a majority vote from classification results of all decision trees. We decided to use RF because it is robust, not susceptible to over fitting and fast classifier. Also, some other classification models can be used instead of a RF, such as a Support Vector Machine model.

This subsystem has several adjustable assumptions on working with a continuous video stream. First, instead of

classifying every frame, it can classify frames with a predetermined frequency to lower working time. Second, once a frame classification results in the "action" class, we can increase the frequency of a classification for a short time, to be able to catch the start of an action. Third, to prevent the wrong transition between subsystems due to misclassifications, the system switches to the action recognition mode only after a predetermined number of the same and consecutive classification results.

In summary, the "pre-action" subsystem allows to compute less by processing fewer frames during the "stand by" mode, and to detect starting and ending point of actions instead of continuously trying to recognize actions in front of it even if no action occurs. In this subsystem, we considered features that could give us information about distance and directions of people motions such as optical flow and boundary box features.

IV. Action Recognition Subsystem

In this section, we present action recognition subsystem. The action recognition subsystem is based on the BoW model and operates only during the "active" mode to determine an action. Each subsection corresponds to a particular step in the BoW pipeline. The BoW model is a supervised model, so it requires training and testing phases, consisting of the following steps[1]. The first step is to process a video stream by extracting features from the input frames. After that, only in the training phase of our model, a codebook of visual words is generated by clustering extracted features. The system passes the codebook and extracted features to the next step. The second step is a computation of histograms of occurrences from video segments, by utilizing the codebook and extracted features. We replace extracted features with corresponding visual words from the codebook, and a histogram stores numbers of the occurrence of these visual words for a video segment. Then all histograms are given as an input to a classification model. And the third step is to make a classification of actions based on the histograms of occurrences.

A. Features Extraction

We extract global and local features. Global features give information from the whole frame and local features describe motions from small patches of interests in a frame. Both types of features are processed separately until the final step of the BoW.

1) Optical flow

We extract HOF as the global feature. HOFs extraction is the same as in Section III - $H = [h_1, \dots, h_{N_{bin}}]$. Each HOF is computed from a single block of a frame. But, this HOF approach doesn't consider a spatial information of a block, which can be helpful for recognition.

We tested two options to encode a spatial position of a block. One option is to concatenate all $n \times m$ HOFs in one frame into one feature vector – $F = [H_1, \dots, H_{n \times m}]$. Another option is to attach the identification value of a block at the end of each HOF. Since blocks positions in the frames are constant - they cover only certain areas, we can give them constant identification numbers from 1 to $n \times m$. In each frame, a block i identification number bl_i can be attached to each HOF extracted from a corresponding block i: $H = [h_1, \dots, h_{N_{bin}}, bl_i]$ for $i \in [1 \dots n \times m]$.

2) Local features

For local features detection, we employed sparse-temporal feature detectors. In recent years, several action recognition researchers published robust feature detectors for videos, such as Harris3D, Cuboid, etc. Unlike the common sparse detectors, these detectors additionally track changes along the time, which allow us to find the changing regions in frames and ignore the background regions. For each input frame, our system detects points of interest and extracts one feature vector from each detected point.

Our system used modifications of FAST and Harris feature detectors - FAST3D detector [12] and Harris3D detector [8]. The FAST3D has the same corner detection algorithm in sparse dimensions as FAST, but it also adds the temporal dimension. Fig. 3 shows examples of detected points. Harris3D is a modification of the Harris corner detection algorithm; it's proven to be an efficient detector in several works [2], [5]. These methods detect a set of points - $P = [p_1, \dots, p_s]$, from a frame, where s is a number of detected points in a frame.

As a descriptor, we decided to use HOG-HOF method. It showed good results in surveys [13], [14]. Around each detected point p_i for $i \in [1 \dots s]$, this descriptor creates a space-time region with the size $r_x \times r_y \times r_z$ and extracts 4-bins histograms of oriented gradients - $Hg = [hg_1, \dots, hg_4]$ and optical flow - $Hf = [hf_1, \dots, hf_4]$ from this region.

Fig. 3. Example of sparse-temporal points using FAST3D (Red dots are detected points). FAST3D detects points only in areas where was movements. In the left image, person was coming closer to the camera, and in the right, person started to move his hand to make some action.

Both resulted histograms are concatenated to one feature vector - $Hl_i = [Hg_i, Hf_i]$. All computed Hl_i are used to make a codebook during training and then, are given to the next step.

B. Codebook Generation

One of the most popular and simple methods to generate the codebook is a k-means clustering. The k-means method clusters set of extracted feature vectors to a K number of clusters, where each cluster represents one visual word. Each feature vector inside a cluster assigns to a corresponding visual word. Generated codebook passed to the next step of the BoW pipeline, along with extracted feature vectors.

There are also several other clustering methods with different pros and cons, and some are considered more efficient in a certain number of applications, such as

[1]Both phases have the same pipeline, except the "codebook generation" part which happens only during a training - after the feature extraction step.

Gaussian Mixture Model (GMM), Hierarchical k-means clustering, etc. All these methods have various parameters that might greatly affect training time and accuracy of the classification. However, in this work, we do not focus on trying different clustering methods, since clustering part has much less effect on the computation time during testing.

Note that our system generates a separate codebook for each type of features.

C. Histogram Computation

The histogram of occurrences is K-dimensional vector $H = [h_1, \dots, h_K] \in R^{D \times K}$, where h_i is a D-dimensional vector. The purpose of histograms is to describe video segments for a later classification. A video segment can be represented as $X = [x_1, \dots, x_K] \in R^{M \times N}$, where x_i is an input feature described by M-dimensional vector and N is a number of features in the video segment.

We compute histograms of occurrences from an input set of feature vectors by using different methods of encoding and pooling from the codebook. After that, computed histograms are given to a classification model.

Since we were aiming to work in the real-time, we processed set of frames as they enter successively. So, each histogram is computed from a part of a video, not the whole one, which can be done using a sliding window. The sliding window has a constant size s_{sw} and moves through a video with a predetermined constant step.

To increase performance, we considered different types of histograms of occurrences to work with a continuous video.

The first type is *the separate histogram computation*. In every position of the sliding window, the system computes a histogram of occurrences from all frames that are inside of the sliding window. Thus, all histograms are based on different periods of time of a video. In this approach, previous information does not affect a current histogram classification decision. This approach heavily depends on the window size, because a short sliding window will not have the necessary information for correct classification, and a long window will conceal important motion changes.

The second type is the *growing histogram computation*. In this case, the system computes the first growing histogram from an initial position of the window as in the separate histogram approach. Then in every next position of the sliding window, the system summarizes a previous growing histogram and a separate histogram from a current position: $h_g(i) = h_s(i) + h_g(i-1)$, $h_g(0) = h_s(0)$, where h_g is a growing histogram and h_s is a separate histogram. So histograms grew consistently. This approach considers progressive temporal changes, but requires a starting and an ending points.

The third histogram computation type uses a predetermined number of histograms layers and it is called the *pyramid histogram computation*. The system constructs a first layer with all separate histograms as in the first approach, and every next layer consists of histograms that computed by a consecutive summation of two neighbor histograms from a previous layer. This type considers both temporal changes and current video segment.

1) Encoding

Encoding is a step to compute a histogram of occurrences, where we connect extracted feature vectors with visual words from a codebook. To calculate a histogram of extracted features from a video segment, the system encodes X, to a set $C = [c_1, \dots, c_N] \in R^{D*K \times N}$, which later pooled to a single vector H.

We considered different encoding methods and chose two following methods: Vector Quantization (VQ) and Super Vector Coding (SVC) [6], [15]. Both methods are popular and represent different approaches on handling visual words. VQ is a standard voting method, where for every feature $x_j, j \in [1..N]$: $C_{j,nr} = 1$ and $C_{j,i} = 0$, $i \neq nr$, where nr is the id of the nearest visual word to x_j. SVC is an extension of VQ, but belong to the Super Vector based methods that use a feature description to extend the size of histograms of occurrences. SVC: $C_{j,nr} = [\alpha C(j), C(j)(x_j - w_{nr})] \in R^{1+M}$ and $C_{j,i} = 0$, $i \neq nr$, where w_{nr} is a vector of the nearest visual word, j – a number of the current feature, $j \in [0..N]$, α is a positive constant, $C(i)$ is a result of the VQ and equals to 1. According to [6], SVC yields good performance and processing speed. Several more efficient encoding methods, such as Fisher Vector, have a better accuracy, but require much more time to compute.

2) Pooling

After encoding features, we summarize a set C to a histogram H. There are two common pooling algorithms: max-pooling and sum-pooling. In the max-poling scheme, we choose maximum value in each column: $h_i = \max(C_{j,i})$, for $\forall j \in \{0..N\}$, and in the sum-pooling, we summarize all values in each column $h_i = \sum_{j=1}^{N} C_{j,i}$. We pass resulted histogram of occurrences H to the next step as an input to a classification model.

Some researchers propose to normalize a histogram of occurrences after computing it [6], because of different numbers of input features in each histogram. But in our findings, there weren't any considerable improvements on using normalization. Also, we can not use normalization methods that compute a general denominator from all histograms in a video, because of the real-time processing, we process histograms as separate entities that come up successively. Instead, we limited the number of detected features, so most of the frames have the same amount of features.

D. Classification

We trained and tested BoW with a supervised Random Forest (RF) classifier. Support Vector Machines (SVM) model is more commonly used in BoW, but RF is considered to be slightly faster in a classification decision. Both classifiers with adjusted parameters show similar performance in many applications.

RF takes vectors H_i for $i \in [0..T]$ with corresponding class values as an input, where T is the total number of histograms, all H_i are histograms of occurrences computed in the previous step. The total number of histograms T is equal to a number of different sliding window positions in all given videos. RF classification result for a single histogram is a single class of an action that happens in the corresponding video segment.

E. Combining Features

We have two different types of features. Often a combination of different features can give better results.

There are a couple of places in the system where we can combine these features [6]: before or after a classifier. We can not combine them earlier because they have a different amount of samples in each frame. Before giving input to a classifier, we can combine both features by concatenating their histogram vectors. The combination after classifier requires to make two separate classifications, which should output vectors of probabilities belonging to each class: $R_i = [R_{i,1}, \dots, R_{i,c}]$, where i - id of a histogram, c - number of classes and $R_{i,j}$ is a probability of histogram i belonging to a class j. So we can combine features by summing exponential functions of resulted class probabilities with respective weights:

$$\text{result} = \max[-\frac{1}{e^{R_{\text{local}} \times w}} - \frac{1}{e^{R_{\text{global}} \times w}}] \quad (1)$$

where w is the weight for local features. This exponential function has a curve that helps to lower small probabilities and emphasize bigger probabilities. Thus, our function will select a class with a high probability in it from any classifier, even if another classifier has a very low confidence in this class.

A result from (1) is a single class that has the maximum probability. Depending on w, the system prioritizes the local or global features. A person close to the camera has a bigger surface to track OF, and, on the contrary, a person at a far distance can be better described by a local feature.

V. Evaluation

The system was evaluated in a conventional laptop with the following system specifications: SSD, 2.4 GHz Intel Core i5, 8 GB RAM. None of the used methods was parallelized nor optimized. The system was implemented in C++ using OpenCV 2.4.9.

We had two goals for our evaluation. The first goal was to find the best combination of parameters for the action recognition subsystem and compare it with related work. The second goal was to find the best approach in the pre-action subsystem and its usefulness for an action recognition.

For evaluation, we used accuracy and frame per second (FPS) metrics. FPS is used to measure processing speed, for the real-time application we need at least 30 FPS. To simulate action recognition from real-time video stream we classified segments of video in each position of sliding window instead of classifying whole video clips. So, each video divided into many labeled segments depending on sliding window size.

A. Dataset

We tested our system in public JPL - First person interaction dataset. This dataset was used in [2] and contains two types of videos with seven different actions: segmented and continuous videos. Each segmented video contains a single occurrence of an action without "pre-action" step, and each continuous video contains one or multiple actions and "pre-action" step. There are five types of close ranged actions: patting, hug, handshake, waving, and punch (harmful action); and two far ranged actions: pointing and throwing an object (harmful action).

For testing the pre-action subsystem, we used the continuous videos from the dataset. These videos are longer and much more diverse: videos contain the pre-action and action with different lengths, some passing people, multiple people in one frame, camera movements, and different actions. There are total 57 videos in this set.

B. Results

The action recognition system was tested using k-fold cross-validation (k=4) on segmented videos dataset. The efficiency and computation time of the system was tested by comparing the following parameters: local features methods, OF parameters, histogram types, histogram encoding methods and features combination methods.

Firstly, we tested different position encoding methods for the sparse HOF method. As shown in Table I simple block id method gave the best results and in further tests, we used this method.

TABLE I: Comparison of Position Encoding Methods for Global Features

Type of position encoding	Accuracy (%)
Without position	72.6
Concatenation	74.2
Block id	76.8

For comparison to our sparse feature extraction methods, we additionally implemented feature extraction methods taken from basic BoW in [2]. This baseline system used following types of features: Dense HOF as a global feature, Cuboid as a local feature detector and XYT as a local feature descriptor. Dense HOF extracts one feature vector per frame, by concatenating Dense HOF of all $m \times n$ blocks in a frame to give us spatial information.

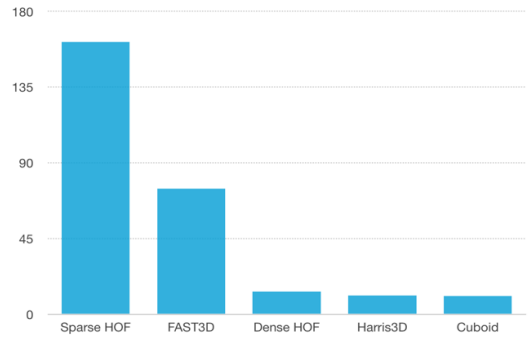

Fig. 4. Speed comparison of features extraction methods (FPS).

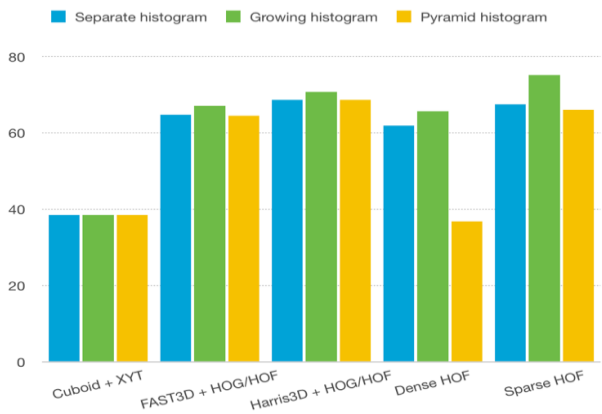

Fig. 5. Comparison of histogram computation methods and feature type by accuracy (%).

We compared the following sets of local features detectors

+ descriptors: Cuboid + XYT, our FAST3D + HOG/HOF and Harris3D + HOG/HOF. Also, we compared the following sets of global features extraction methods: Dense HOF and our Sparse HOF. We measured processing time and classification accuracy of these methods. The processing time for all methods is roughly given in the Fig. 4. As we can see, proposed sparse OF and FAST3D methods considerable outmatched baseline methods. Dense HOF speed can be further optimized by using fewer input points, but, in this case, accuracy would fall.

Together with feature extraction methods, we compared three different proposed methods to compute a histogram of occurrences: separate, growing and pyramid. Results of feature extraction methods with histogram computation methods are presented in the Fig. 5. The Sparse HOF gave better results, mostly because it is computed from special points that are more distinguishable than a grid of points in Dense HOF. The Harris3D showed the best performance among local features; however, its processing time is much slower than in FAST3D.

Next, we compared VQ and SVC encoding methods for a histogram of occurrences computation. The VQ method is simple and was efficient in our case, SVC showed slightly worse results and computation time was much higher than for VQ.

For the features combination methods, we used the following combinations of local and global features: Cuboid + XYT with Dense HOF, FAST3D + HOG/HOF with Sparse HOF and Harris3D + HOG/HOF with Sparse HOF. Fig. 6 shows results of this comparison, where 'Concatenation' stands for features combination method that concatenate histogram of occurrences vectors before classification and 'Merge' is a features combination method that merge results of two separate classifiers. All tests used the “growing” type histogram as it previously showed the best results.

From Fig. 6 we can see that both feature combination methods did not get considerably better results in comparison to the separate use of features. Also, local features could not get better results than HOF (Fig. 5) in most videos and had a lot of misdetections during a camera shudder. Most of the actions required to have contact with a camera, which caused a shudder/fall of the camera with various duration and speed. These kinds of noises make harder to detect and extract local feature points in the foreground. Thus, the overall usefulness of these local features in the real environment is under a question.

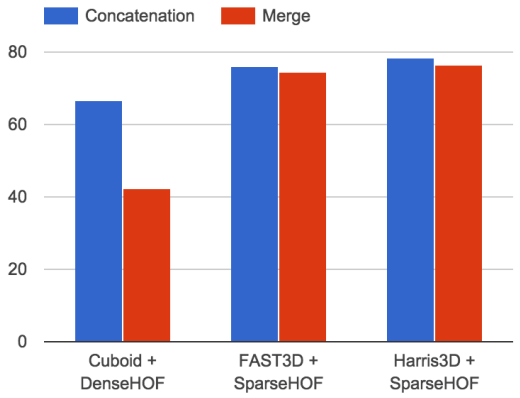

Fig. 6. Comparison of encoding and features combining methods (%).

Table II shows confusion matrix of actions recognition. Some of the actions, such as "punching", had poor recognition because these actions were executed fast and did not have much recorded time that results in the lack of samples in the dataset. Action “pet” and “hug” were quite similar, mostly because they were made in the close proximity to a camera and the camera couldn’t get the whole view of a human in front of it: camera can’t see hands and head of a human that are standing too close. Similarly, long-ranged actions (“throw” and “point”) had some misclassifications.

TABLE II: CONFUSION MATRIX (%) OF ACTIONS AND NUMBER OF SAMPLES IN EACH ACTION

	Handshake	Hug	Pet	Wave	Point	Punch	Throw	# of samples
Handshake	71.7	0	13.3	0	0	0	15	60
Hug	4.6	77.9	17.6	0	0	0	0	131
Pet	11.3	53.6	34	1	0	0	0	97
Wave	20	0	10	35	15	0	20	20
Point	0	0	0	0	99.6	0	0.4	236
Punch	30	0	35	0	0	30	5	20
Throw	6.7	0	0	2.2	35.6	0	55.6	45

In summary, we experimented with a different set of feature extraction methods and other parameters of our system in order to make it work in the real-time from a video stream. By choosing the best combination, we considered a trade-off between accuracy and processing speed. For our goals, the best parameters combination of the action recognition subsystem would be the concatenation of local features, which are computed using FAST3D and HOG-HOF methods, and global feature Sparse HOF.

C. Pre-action System

The pre-action system was tested using k-fold cross-validation (k=4) in the continuous videos of JPL dataset. All videos were divided into two classes: pre-action and action. We tested two features extraction methods: sparse HOF and boundary box. Results are shown in Table III. Here we should make a trade-off between speed and accuracy: boundary box method gave better accuracy, but OF was much faster.

TABLE III: COMPARISON OF SPEED AND ACCURACY OF PRE-ACTION FEATURE EXTRACTION METHODS

Methods	Speed (FPS)	Accuracy (%)
Sparse HOF	67.3	76.2
Boundary box	22.7	88.0

Finally, we evaluated the usefulness of proposed pre-action subsystem. We compared two approaches that can be used to work in continuous videos: continuous action recognition approach and our approach that uses the pre-action

subsystem. The continuous action recognition approach is a regular approach, where a system continuously recognizes actions from a video stream, and also add "nothing" class when there is no action occurs.

Both approaches used almost the same parameters of action recognition subsystem, except type of histogram computation method. Pre-action subsystem approach used "growing" histogram computation method because this method requires to have the starting and ending points. And continuous action recognition approach does not compute a start and end of the actions; therefore, it used "separate" histogram computation method.

We trained both approaches on the segmented dataset and several video segments that do not contain any action. The results showed 60.1% accuracy for our approach and 32.3% for continuous recognition.

It can be concluded that the pre-action subsystem is a good approach to integrate with a real-time action recognition system that works with a continuous video stream. And it has several parameters that can be tuned depending on the situation: frequency of classification, a maximum number of same consecutive results, feature extraction method, and classification model parameters.

VI. Conclusion

In this work, we designed and implemented the system for real-time action recognition from FPV continuous video stream. The classic BoW model was adopted for an action recognition and modified by using different sparse features instead of dense features, methods to process continuous stream and other parameters. We experimentally evaluated the performance and confirmed the improvement in accuracy and processing speed. Additionally, we proposed the pre-action subsystem that helps to work with continuous video stream by detecting the starting and ending points of actions. Thus, the system recognizes actions only when it is necessary. The evaluation showed improvement in the classification rate of continuous action recognition with the pre-action subsystem.

In the future works, it might be a good idea to consider uses of depth information, because many robot systems employ 3D cameras. The depth information can help to improve the feature detection and can give additional features, such as human joints positions and 3-D optical flows. Another direction for the future research might be analysis of videos from more dynamic cameras, where we should take into account frequent camera motions.

References

[1] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *Proc. ECCV Workshop on Statistical Learning in Computer Vision*, pp. 1–22, 2004.

[2] M. S. Ryoo and L. Matthies, "First-person activity recognition: What are they doing to me?" in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2013.

[3] A. Fathi, J. K. Hodgins, and J. M. Rehg, "Social interactions: A first-person perspective," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 1226-1233.

[4] B. R. Fransen, W. E. Lawson, and M. D. Bugajska, "Integrating vision for human-robot interaction," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2010, pp. 9-16.

[5] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, "Learning realistic human actions from movies," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2008.

[6] X. Peng, L. Wang, X. Wang, and Y. Qiao, "Bag of visual words and fusion methods for action recognition: Comprehensive study and good practice," 2014.

[7] J. R. R. Uijlings, I. C. Duta, N. Rostamzadeh, and N. Sebe, "Realtime video classification using dense HOF/HOG," in *Proc. International Conference on Multimedia Retrieval*, Association for Computing Machinery, 2014.

[8] I. Laptev and T. Lindeberg, "Space-time interest points," in *Proc. International Conference on Computer Vision*, 2003, pp. 432–439.

[9] E. Rostenand and T. Drummon, "Machine learning for high-speed corner detection," in *Proc. the 9th European Conference on Computer Vision*, Springer Science Business Media, 2006, pp. 430–443.

[10] T. Tuytelaars and K. Mikolajczyk, "Local invariant feature detectors: A survey," *Foundations and Trends® in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177-280, Jan. 2008.

[11] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. CVPR*, 2005, pp. 886–893.

[12] S. Koelstra and I. Patras, "The fast-3d spatio-temporal interest region detector," in *Proc. 10th Workshop on Image Analysis for Multimedia Interactive Services*, Institute of Electrical & Electronics Engineers (IEEE), May 2009.

[13] P. Bilinski and F. Bremond, "Evaluation of local descriptors for action recognition in videos," *Computer Vision Systems*, Springer Science Business Media, pp. 61–70, 2011.

[14] H. Wang, M. M. Ullah, A. Klser, I. Laptev, and C. Schmid, "Evaluation of local spatio-temporal features for action recognition," in *Proc. the British Machine Conference*, Sep. 2009.

[15] X. Zhou, K. Yu, T. Zhang, and T. S. Huang, "Image classification using super-vector coding of local image descriptors," in *Proc. the 9th European conference on Computer Vision*, Springer Science Business Media, pp. 141–154, 2010.

[16] J. K. Aggarwal and M. S. Ryoo, "Human activity analysis," *ACM Computing Surveys (CSUR)*, vol. 43, no. 3, pp. 1-43, Apr. 2011.

[17] K. M. Kitani, T. Okabe, Y. Sato, and A. Sugimoto, "Fast unsupervised ego-action learning for first-person sports videos," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2011, pp. 3241-3248.

[18] R. Poppe, "A survey on vision-based human action recognition," *Image and Vision Computing*, vol. 28, no. 6, pp. 976–990, Jun. 2010.

M. Maximov received the B.S. degree in information technology from North-Eastern Federal University, Russia and M.E. degree in human-computer interaction & robotics from Korea University of Science and Technology, Korea, in 2012 and 2015, respectively. His research interests include computer vision, text processing and machine learning.