2017 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI 2017)
November 16 - 18, 2017, Daegu, Korea

Wearable Gesture Control of Agile Micro Quadrotors

Yunho Choi, Inhwan Hwang, and Songhwai Oh

Abstract— **Quadrotor unmanned aerial vehicles (UAVs) have seen a surge of use in various applications due to its structural simplicity and high maneuverability. However, conventional control methods using joysticks prohibit novices from getting used to maneuvering quadrotors in short time. In this paper, we suggest the use of a wearable device, such as a smart watch, as a new remote-controller for a quadrotor. The user's command is recognized as gestures using the 9-DoF inertial measurement unit (IMU) of a wearable device through a recurrent neural network (RNN) with long short-term memory (LSTM) cells. Our implementation also makes it possible to align the heading of a quadrotor with the heading of the user. Our implementation allows nine different gestures and the trained RNN is used for real-time gesture recognition for controlling a micro quadrotor. The proposed system exploits available sensors in a wearable device and a quadrotor as much as possible to make the gesture-based control intuitive. We have experimentally validated the performance of the proposed system by using a Samsung Gear S smart watch and a Crazyflie Nano Quadcopter.**

I. Introduction

A quadrotor is a popular platform for an unmanned aerial vehicle (UAV) and consists of four rotors on its four corners of the cross frame. It is controlled by changing the rotational speed of each rotors. It is structurally simple, which means that it can be built and maintained by amateurs. It is also highly maneuverable and is capable of hovering, vertical takeoff and landing (VTOL) [1]. For these reasons, quadrotors have been popularized for UAV research and applications such as rescue, surveillance, commercial, and personal entertainment. To take advantage of its maneuverability and maximize its utility, a proper remote-control method is essential.

Existing remote-controllers for quadrotors have a number of limitations. They are usually equipped with two joysticks. One joystick is used to control the roll and pitch of a quadrotor and the other is used to control the thrust and yaw of a quadrotor. However, this method incurs many inconveniences due to its unnatural use of joysticks. First of all, it requires user's both hands for handling two joysticks. It makes the user "short-handed" literally and hinders the user from carrying out other tasks, e.g. taking pictures, watching a first person view flight video with a smartphone, or even controlling another quadrotor. In addition, since a quadrotor usually moves around with respect to its body frame when remote-controlled, the user cannot control it intuitively if he or she is not aligned to the body frame of the quadrotor. In other words, the user should control a quadrotor while considering the body frame of a quadrotor, not his or her own frame. As a result, this contrived method requires lengthy training before a user gets familiar with the controller. Lastly, the controller is also too big to carry around.

Yunho Choi, Inhwan Hwang and Songhwai Oh are with the Department of Electrical and Computer Engineering and ASRI, Seoul National University, Seoul 08826, Korea (e-mail: {yunho.choi, inhwan.hwang}@cpslab.snu.ac.kr and songhwai@snu.ac.kr).

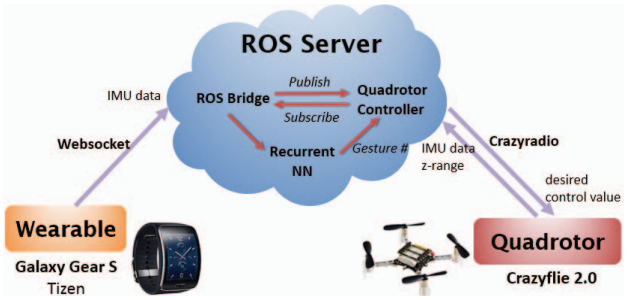


Fig. 1. An overview of the proposed system. Communication methods and messages between a Robot Operating System (ROS) server, a wearable device and a quadrotor are specified. Three components in the ROS server are ROS nodes and an arrow shows the direction of ROS message flows.

Thus, we propose a new remote-control system for a quadrotor UAV using a wearable device in a watch form by controlling a quadrotor based on recognized gestures of a user. Wearing a user-friendly device on a single wrist gives much freedom to the user. A smartphone can also be used but a smart watch has a more convenient form factor as it makes both hands free. Since the user does not need to hold any device when wearing a watch device instead of a smartphone, we can make use of gestures with hands for control commands. Just as a dog owner easily manages a dog that is trained to follow the owner's gesture commands, controlling a quadrotor with the user's hand is a more natural choice.

In order to control a quadrotor using gestures which are easy and intuitive, our system fully utilizes available sensors such as inertial measurement units (IMUs) in both the quadrotor and the wearable device, each consisting of an accelerometer, a gyroscope and a magnetometer, and a laser range sensor attached under the quadrotor. An overview of the proposed system is presented in Figure 1. Specifically, we collect sequential IMU data of several predefined gestures using a wearable device and train a deep neural network to classify gestures. To deal with sequential sensor data, we use a recurrent neural network (RNN) with long short-term memory (LSTM) [2] cells. In the test phase, measurements from the IMU of a wearable device are sent to the Robot Operating System (ROS) [3] server. Then, by feed-forwarding the measurements to the trained deep neural network, the system recognizes which gesture the user has performed. Considering the performed gesture together with

978-1-5090-6064-1/17/$31.00 ©2017 IEEE

the data from the IMU of a quadrotor and a laser range sensor underneath the quadrotor, the system decides what command to give to the quadrotor. The system can also resolve the frame misalignment problem. From the magnetometer data of the wearable device and the quadrotor, we can compute the difference of the heading between the user and the quadrotor. Then we can align them by spinning the quadrotor to align with the user's frame and regulate the difference to zero.

In this paper, we propose a remote-control system using a smart watch device, which effectively controls a quadrotor with intuitive gestures. Fusing multiple low-cost sensors including IMUs in a smart watch device and a quadrotor, and a laser range sensor of the quadrotor, the proposed method successively enables gesture recognition and quadrotor control. The proposed system solves limitations of conventional controllers such as non-intuitive control and inconvenience of holding a big controller with two hands, by exploiting intuitive control gestures with free hands. With the proposed system, we can easily command various motions to a quadrotor, including taking off, landing, and hovering as well as basic commands available in traditional controllers.

The remainder of this paper is organized as follows. In Section II, we discuss related work in quadrotor control and gesture recognition. Then, we propose a new control scheme for a quadrotor and a gesture recognition network which classifies the gestures needed for the control scheme in Section III and IV, respectively. Implementation details are given in Section V. In Section VI, we validate our system from experiments with a Samsung Gear S smart watch and a Crazyflie Nano Quadcopter which is a palm-sized micro quadrotor that requires smaller controller latency due to its small inertia.

II. Related Work

There have been a number of quadrotor control interface designs using various sensors. First, there are approaches where the position of a quadrotor is controlled autonomously without the manipulation of a user under GPS-denied environments. In early days, tracking a quadrotor using a camera was a way to localize the quadrotor in a restricted environment [4]. Simultaneous localization and mapping (SLAM) implemented with a monocular camera can be used to calculate the pose of a quadrotor which can be used for control [5]. In [6] and [7], an external localization system, such as a Vicon motion capture system, was used to control a quadrotor. In [8], a Kinect sensor was used to estimate the altitude of a quadrotor. These systems in common estimate the pose of a quadrotor for control. Without the pose information, optical flow can be used to estimate the state of a quadrotor. In [9] and [10], a camera attached to a quadrotor was used to calculate optical flow and control the quadrotor. These studies described above have three problems. They require either an expensive system like a Vicon motion capture system, computationally heavy algorithms like SLAM, or a restricted field of view of cameras or Kinect sensors.

There are different approaches where a human user controls a quadrotor and the joystick-based controllers are replaced with more intuitive control methods. A camera [11] or a Kinect sensor [12] is used recognize the user's motion and a command is given to a quadrotor. Also, there are gaze-based control approaches which control a quadrotor according to the gaze of the user obtained from an eye tracking device [13]. However, vision-based gesture recognition methods are not low-cost and have restrictions such as its limited field of view. Gaze-based methods have the same problem and its accuracy is insufficient for indoor flight. Thus we propose an alternative method which makes use of accelerometer and gyroscope data from a wearable device to recognize and classify user's gestures. It is a low-cost solution with no space restriction and also accurate.

For sensor-based gesture recognition, RNN structures have been dominantly used [14], [15]. Since the earlier RNN suffered from long-term dependency problem [16], their performance had limitations. But the LSTM architecture was proposed to resolve this problem and showed good performance in remembering information for a long duration [2]. The core ideas of LSTM are a memory cell maintaining so-called *cell state* over time and nonlinear gate units which adjust information flow in and out of the cell state. It has been actively employed to many applications including speech recognition, translation, and language modeling and also resulted in many variants, such as the gated recurrent unit (GRU) [17]. In the proposed system, an RNN structure including LSTM cells is exploited to classify gestures.

III. Control Scheme

When a user wearing a wearable device performs a gesture, we need to provide an appropriate command to a quadrotor based on the recognized gesture from the gesture recognition network. These series of processes can be expressed in a form of a finite state machine (FSM) and shown in Figure 2. We call this diagram a control scheme. In the proposed system, a user can shift the state, which will also be referred to as the mode, of a quadrotor by taking a gesture according to the control scheme. The user can also control the attitude and height of the quadrotor with simple tilting motions and gestures. Along the scheme, the user can take off the quadrotor and make it hover at an initial height of 0.5 m by taking the *Takeoff* gesture in the initial *Ground* state. Likewise, the user can pilot a quadrotor, i.e. control the attitude and hover height of the quadrotor. In the *Heading Aligned* mode, the heading of the quadrotor is aligned to the heading of the user. The *Come* gesture makes the quadrotor move toward the user. The *Land* gesture makes the quadrotor slowly land on the ground. Furthermore, there is no state change if an invalid gesture or a gesture out of context is taken.

In the *Pilot* mode, the attitude control is done through simple tilting motions, while the height control exploits the *Height Up* and *Height Down* gestures. Specifically, the attitude control is divided into controlling the roll and pitch of the quadrotor. If the user tilts the wearable device so that the roll of the wearable device exceeds $\pm 25^{\circ}$, then the quadrotor rolls to tilt at a proportional angle and consequently drifts

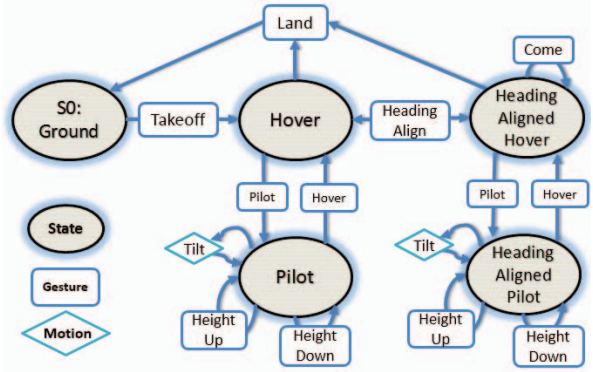


Fig. 2. A basic control scheme expressed in a finite state machine (FSM) form. Each ellipse, box and rhombus represent a state, a gesture command, and a motion command respectively. In the *Pilot* mode we can control the roll, pitch and height of the quadrotor. *Height Up* gesture raises and *Height Down* gesture lowers the height of the quadrotor by 10 cm. *Heading Align* gesture makes the heading of the quadrotor is aligned to the heading of the user.

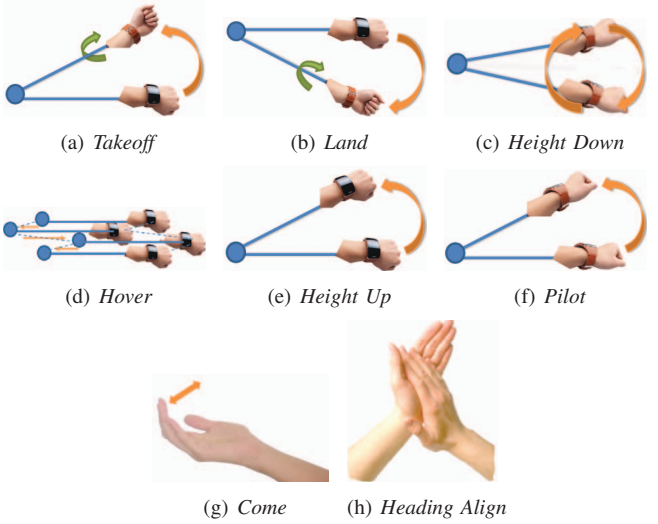


Fig. 3. A simplified diagram of gestures used for our system. Each line and circle represent an arm and an elbow, respectively. Here, the arm is from the wrist to the elbow. Green arrow represents rotational movements of which the axis of rotation is the arm. Gestures are mapped to commands as follows: (a) *Takeoff*. (b) *Land*. (c) *Height Down*. (d) *Hover*. (e) *Height Up*. (f) *Pilot*. (g) *Come*. (h) *Heading Align (Clap)*.

to y direction, side-to-side. It works through the on-board proportional-integral-derivative (PID) control. In case of the pitch of the wearable device exceeding $\pm 25^\circ$, it works the same way except that it pitches and drifts to x direction, front-to-back. Moreover, in the *Heading Aligned Pilot* mode, the yaw of the quadrotor is PID controlled, targeting the yaw value of the wearable device. Thanks to this new feature, the tilting directions of the wearable device and the quadrotor are always the same when controlling the attitude of the quadrotor and it results in more intuitive manipulation.

To control the height of the quadrotor in every states, we use the sensor value from the laser range finder underneath the quadrotor. A PID controller regulates the height to the desired value. All the details of PID controls which stabilize the quadrotor are given in Section V.

For our control scheme, we need eight gestures to map into the control scheme, which are *Takeoff*, *Land*, *Pilot*, *Hover*, *Height Up*, *Height Down*, *Heading Align*, and *Come*. The proposed gestures are shown in Figure 3. Since in the *Pilot* mode the quadrotor drifts to some direction if the roll or pitch of the wearable device exceeds $\pm 25^\circ$, gestures involved with the *Pilot* mode such as *Hover*, *Height Up*, and *Height Down* must not make the roll and pitch exceed $\pm 25^\circ$, i.e., in order to prevent unintended drifts. While rotational movements of the user's arm can make the roll and pitch of the wearable device greater than $\pm 25^\circ$, these gestures do not involve rotational moves.

IV. Gesture Recognition Network

In this section, we propose the RNN structure as shown in Figure 4 to recognize gestures and explain how we model and train the network. With the RNN network, We classify the sequential data of eight command gestures we predefined and an additional "no-op" (no operation) gesture induced from negative data, thus nine gestures in total. From the result of the RNN classifier, we recognize a gesture and give a correspondent control command based on the control scheme presented in Figure 2.

The objective of our network is basically classifying the user's gesture as one of nine gestures from the sequential input of accelerometer and gyroscope data. As shown in Figure 4, the input sensor data is a matrix of fixed size $3 \times L \times 2$ for three axes data from two sensors, where L is the maximum sequence length. In our implementation, L is set to 50 since the length of all gestures does not exceed 50, which is equivalent to two seconds in duration. The sequence length L of a gesture is the interval between the start and the end of motion which are obtained by thresholding the moving average acceleration value of the wearable device.

We employed an RNN structure combining convolutional layers and LSTM cells. Convolutional filters learn temporal features of sensor data and then LSTM will learn temporal relations between those features. This structure makes it possible to remember past information. To learn inter-axial features, the network first passes input data into two convolutional layers. The size of convolution filters must be properly determined. If it is too large, sequential features will be lost, while temporal features are not captured if it is too small. The numbers used for filter sizes and strides are given in Figure 4 and they are determined empirically. After the convolutional layers, a fully connected layer is applied in the feature dimension and the results are split along the time dimension to make sequential inputs to LSTM cells. Sequential information is preserved through the whole process. The LSTM layer consists of two stacked LSTM cells to allow greater model complexity.

The final output of the LSTM layers is the output of the last LSTM cell, the L-th output. We call that cell the last relevant cell. The output of the last relevant cell is passed into a fully connected layer and returns a vector whose size is the number of predefined gestures. From this vector we

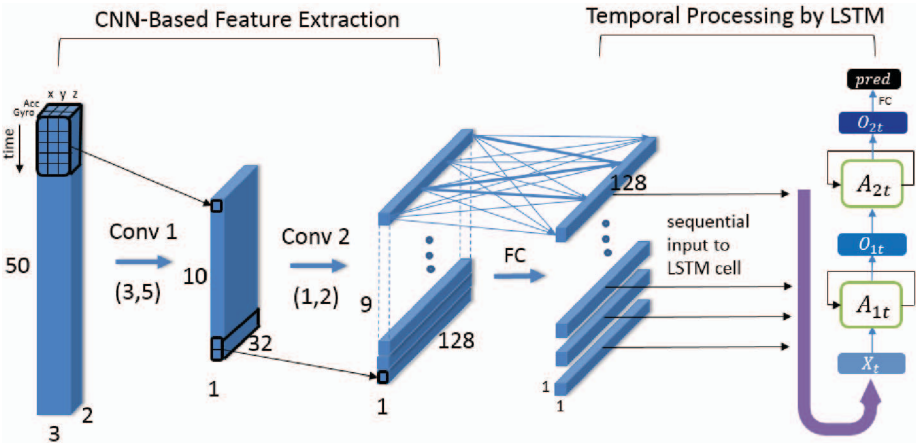


Fig. 4. The recurrent neural network structure used to recognize the gestures. Data dimensions and filter sizes are shown in the figure. Three-axes data of two sensors (accelerometer, gyroscope) are fed in the network. First, they pass through two convolutional layers and a fully connected layer. No padding for convolutional layers is used. Strides for two convolutional layer are five and one along time axis, respectively. A sequentially sliced vector is fed into two layers of stacked LSTM cells. The output of the last relevant cell according to the gesture sequence length is passed into a fully connected layer and results in a gesture prediction vector whose size is the number of gestures (nine in our case).

classify which gesture the user has conducted.

We train the network end-to-end with a softmax cross entropy loss function. In the test phase, when real-time sensor data is sent from the wearable device to an ROS server node, we detect the start and end point of the motion in the same way as in the case of training data. Every time a motion is detected, sensor data for two seconds from the start point of the motion is fed into the network as input. The network, which has weights restored from the trained network, then feed-forwards the input and outputs the classification result.

When training the network, we also make use of negative data which is sampled between gesture motions. They are labelled as the *no-op* gesture and trained together with the other eight gestures. In the test phase, it plays an important role since we can reject misrecognized motions which are performed by the user unconsciousness. This not only allows us to classify as one of the valid gestures, but also enables to detect invalid motions resulting in more safe operation.

V. Implementation Details

A. Wearable Device

For a wearable device, a Samsung Gear S is used. It has a 9-DoF IMU and a WiFi module. Therefore, through the Tizen web application in HTML/JavaScript we developed, we can send IMU data to the server through a web socket. The ROS server, which communicates with a quadrotor robot, gets the sensor data from the wearable device through the web socket in a ROS bridge [18] node. The sampling frequency of the IMU in Gear S is 25 Hz and the server gets the sensor data also in 25 Hz.

B. Training Data Collection

Training data of the predefined gesture motions in Figure 3 is collected. The duration of each gesture does not exceeds 2 s, which bounds the sequence length by 50 due to the sampling frequency of 25 Hz. Here, a user's basic posture is assumed as a posture in which the arm is lifted slightly and horizontally just as when we watch the wristwatch. Intuitive gestures and gestures in common use are chosen for command gestures. Gestures can be replaced with other gestures assuming they are distinguishable. Seven people participated in gathering IMU data of command gestures with 50 repetitions for each of eight gestures. Data augmentation was done with small perturbation in time, adding a Gaussian noise, smoothing and scaling. The overall data set is divided into 80% for a training set (12,205 samples) and 20% for a test set (3,070 samples).

The post-processing and data set generation are done in MATLAB. To figure out the motion start time and end time from the raw sensor data, we threshold the moving average of four adjacent device acceleration values.[1] Then we abort sequences whose lengths are shorter than 25 and sequences in which device acceleration values never exceed 4.0 $\mathrm{m/s^2}$. Normalization is done to sensor values for each sensor. The mean and standard deviation value computed from the training set for each sensor are used to normalize sensor measurements during testing. From the gesture sequence length and the deep neural network model, the LSTM cell number of the last relevant cell can be calculated. Negative data is sampled between gesture motions are labelled as *no-op*'s and trained together with the other eight gestures. To balance the learning process, we have matched the number of negative samples to the number of samples for a single gesture.

C. Deep Neural Network Training

The deep neural network proposed in the system is implemented in Python using the Tensorflow library [19]. A hyperbolic tangent function is chosen as an activation function after two convolutional layers. For training, the Adam optimizer is used and the learning rate is set to 0.001. Also, dropouts in input and output of LSTM are used with the

[1]The threshold value is empirically determined to 3.5 $\mathrm{m/s^2}$ in our system.

dropout ratio of 0.5. We have used a softmax cross entropy loss function to train the network end-to-end. The mini-batch size is set to 512 and the network is trained over 25 epochs. The training result are given in Section VI.

D. Quadrotor

The quadrotor platform used in the system is a Bitcraze Crazyflie Nano Quadcopter, which is one of the smallest quadrotor in the market. It measures 92 mm from a propeller to a propeller and weighs only 27 g. Its small form factor makes a good research platform which can fly near people in indoor environments. However, its small inertia requires a controller with a capability of reacting with very small latency. Bitcraze also provides an extension deck which connects a laser sensor (VL53L0X) to a Crazyflie quadrotor. It is one of the smallest time-of-flight (ToF) ranging sensor. In addition, its low cost and low power consumption make it the most appropriate laser range finder to work with Crazyflie quadrotors. It can measure the distance to the floor up to 2 m with a sampling rate up to 50 Hz.

The server in the proposed system controls and communicates with Crazyflie 2.0 with the ROS Crazyflie driver [20] via the USB dongle named Crazyradio. ROS nodes in our server consist of an ROS bridge node, a Crazyflie server node implemented in C++ and a gesture recognition node implemented in Python. The Crazyflie server node receives the sensor data from a wearable device and a quadrotor, and sends them to the other nodes. The gesture recognition node, which imports the Tensorflow library and the gesture recognition network with the restored trained weights, classifies gestures and sends it to the controller node. Then the controller node sends commands to a Crazyflie quadrotor according to the recognized gesture and its state based on the control scheme.

The control input that the controller node sends to the Crazyflie quadrotor consists of roll, pitch, yaw, and thrust values. The roll, pitch and yaw values are the desired quadrotor attitude values which are determined from the attitude of the wearable device in the *Pilot*, *Heading Aligned Hover* and *Heading Aligned Pilot* states and are zero in the other states. The internal PID controller of the Crazyflie then regulates the difference between the desired attitude values and the current attitude values estimated by the IMU sensor fusion. The thrust value is obtained by another PID controller that we implemented in the controller node. It regulates the difference between the laser range sensor value and the desired height value of the Crazyflie to zero.

For the *Heading Aligned* mode, the heading of the quadrotor is PID controlled targeting the heading of the wearable device. To do so, first of all we set the zero-yaw direction of the Crazyflie to be equal to that of the wearable device. Since the zero-yaw direction of the Crazyflie is the direction it is headed when it boots, we can set it easily. After setting the zero-yaw direction, we perform simple computations so that the yaw ranges from 0° to 360° clockwise to match the yaw range of Gear S we are targeting. Then, the error for yaw PID control can be computed. In order to determine which of the clockwise and counterclockwise directions is closer to the target heading and rotate the quadrotor in that direction, we compute the remainder when dividing the error by 360 and reset the error depending on whether the remainder is greater than 180°.

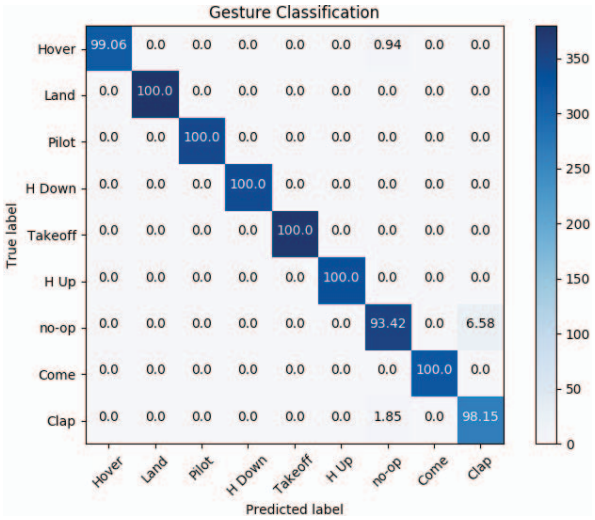


Fig. 5. A confusion matrix obtained from the test set. Each label stands for the alias of each gesture. Proportions of predictions for each ground truth gestures are given as percentages. Colors in the rightmost bar represent the number of data.

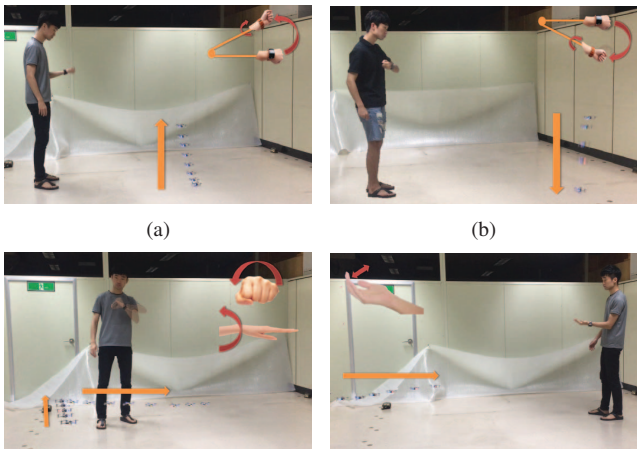
VI. Results

The gesture recognition network is trained well and the training accuracy converges to 99.9% and the test accuracy is about 98.9% with the collected data set. Figure 5 shows a confusion matrix of classification accuracy from the test set. The network correctly classifies gestures except that it slightly confuses with two gestures, *no-op* and *Clap*. It implies that the *Clap* gesture accompanies small movements such that the network gets confused with small noisy movements in *no-op*.

Since our objective is to recognize a gesture of a user and control a quadrotor based on the recognized gesture, the gesture recognition accuracy and latency at the test phase when the user moves in real time are much more important. We have validated our system from experiments with a Gear S smart watch and a Crazyflie quadrotor as shown in Figure 6. The gestures of the user and the correspondent movements of the quadrotor are illustrated in Figure 6. At the test phase, we can filter out unintentional motions from unconscious moves consistently as *no-op* gestures. It takes about 3 ms on average for a 2.0 GHz quad core laptop to feed-forward the sequential data and get a classification result. Hence, our system has shown the sufficient accuracy and latency to be exploited to control a micro quadrotor with gesture detection in real-time.

VII. Conclusions

Using low-cost IMUs in a wearable device and a micro quadrotor, and a laser range sensor attached to the bottom of the quadrotor, the proposed gesture control method has



(a) (b) (c) (d)

Fig. 6. Overlaid snapshots from the experiment. In each figure, the gesture that the user is performing and the corresponding movement of the quadrotor are visualized. (a) Takeoff. (b) Land. (c) Pilot (Rolling, Pitching). The user is making the quadrotor to roll and pitch by tilting his arm. (d) Come

successively made it possible to resolve the problem of the conventional controller and gives a convincing alternative which can be applied to control a micro quadrotor. To recognize our control gestures in the server, accelerometer and gyroscope data from the wearable device were sufficient. By exploiting an RNN structure with LSTM cells for real-time gesture recognition, our gesture recognition network in the server classifies user's gestures with a sufficient accuracy and latency. Hence, a user wearing a smart watch can control a quadrotor easily by taking intuitive gestures which are mapped to movements of the quadrotor in accordance with the control scheme. The proposed system can align the heading of a quadrotor with the heading of a wearable device thanks to the IMUs in both, further improving the intuitiveness of control. It makes the quadrotor head towards the same direction as the wearable device with reference to the direction of the earth's magnetic field. The proposed system also has lots of room for extension such as additional gestures to support more actions of the quadrotor, using a smartphone as a server, and adjusting control commands according to the intensity of a gesture induced from the magnitude of the LSTM cell output.

ACKNOWLEDGMENT

This work was supported by SNU-Samsung Smart Campus Research Center (No. 0115-20160031).

REFERENCES

[1] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," in *Proc. of the AIAA Guidance, Navigation and Control Conference*, Aug. 2007.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA), ser. Open-Source Software workshop*, May 2009.

[4] E. Altug, J. P. Ostrowski, and R. Mahony, "Control of a quadrotor helicopter using visual feedback," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2002.

[5] S. Weiss, D. Scaramuzza, and R. Siegwart, "Monocular-slam–based navigation for autonomous micro helicopters in gps-denied environments," *Journal of Field Robotics*, vol. 28, no. 6, pp. 854–874, Oct. 2011.

[6] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin, "Learning quadrotor dynamics using neural network for flight control," in *Proc. of the IEEE Conference on Decision and Control (CDC)*, Dec. 2016.

[7] B. Landry, R. Deits, P. R. Florence, and R. Tedrake, "Aggressive quadrotor flight through cluttered environments using mixed integer programming," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2016.

[8] J. Stowers, M. Hayes, and A. Bainbridge-Smith, "Altitude control of a quadrotor helicopter using depth map from microsoft kinect sensor," in *Proc. of the IEEE International Conference on Mechatronics (ICM)*, Apr. 2011.

[9] F. Kendoul, I. Fantoni, and K. Nonami, "Optic flow-based vision system for autonomous 3d localization and control of small aerial vehicles," *Robotics and Autonomous Systems*, vol. 57, no. 6, pp. 591–602, Jun. 2009.

[10] V. Grabe, H. H. Bülthoff, and P. R. Giordano, "On-board velocity estimation and closed-loop control of a quadrotor uav based on optical flow," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2012.

[11] A. Shetty, A. Shinde, J. Patel, N. Panchal, *et al.*, "Gesture controlled quadcopter," *Imperial Journal of Interdisciplinary Research*, vol. 2, no. 5, pp. 1289–1291, Apr. 2016.

[12] A. Mashood, H. Noura, I. Jawhar, and N. Mohamed, "A gesture based kinect for quadrotor control," in *Proc. of the IEEE International Conference on Information and Communication Technology Research (ICTRC)*, May 2015.

[13] B. H. Kim, M. Kim, and S. Jo, "Quadcopter flight control using a low-cost hybrid interface with eeg-based classification and eye tracking," *Computers in biology and medicine*, vol. 51, pp. 82–92, Aug. 2014.

[14] K. Murakami and H. Taguchi, "Gesture recognition using recurrent neural networks," in *Proc. of the ACM Conference on Human Factors in Computing Systems (CHI)*, Apr. 1991.

[15] G. Bailador, D. Roggen, G. Tröster, and G. Triviño, "Real time gesture recognition using continuous time recurrent neural networks," in *Proc. of the ICST International Conference on Body Area Networks*, Jun. 2007.

[16] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[17] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proc. of the ACL Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Oct. 2014.

[18] C. Crick, G. Jay, S. Osentoski, B. Pitzer, and O. C. Jenkins, "Rosbridge: Ros for non-ros users," in *Proc. of the International Symposium of Robotics Research (ISRR)*, Dec. 2011.

[19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint: 1603.04467*, Mar. 2016.

[20] W. Hoenig, C. Milanes, L. Scaria, T. Phan, M. Bolas, and N. Ayanian, "Mixed reality for robotics," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015.