AIAA SciTech Forum
January 3-7, 2022, San Diego, CA & Virtual
AIAA SCITECH 2022 Forum

10.2514/6.2022-2031

Online Action Change Detection for Automatic Vision-based Ground Control of Aircraft

Qingze Huo[*], Yifeng Shi [†] and Chang Liu [‡]
Cornell University, Ithaca, NY 14853

Vahid Tarokh[§]
Duke University, Durham, NC 27708

Silvia Ferrari[¶]
Cornell University, Ithaca, NY 14853

Classical action recognition algorithms require the user to pre-select the time window for the action by clipping the video or choosing the initial and final time frames. Recently, new deep learning algorithms have been developed to detect a key action from an untrimmed video. However, they are unsuitable for temporally segmenting continuous action sequences and are too computationally expensive for implementation on autonomous systems. This paper presents a fast and accurate online action change detection framework. Given a streaming RGB video, the algorithm is used to detect any changes in the sequence. If a new action is discovered, the action recognition module will be applied to classify the action. Compared to existing methods, this two-stage approach reduces computational cost by not applying the recognition algorithm on every time window and improves classification accuracy by locating the starting time step of each action. In a simulated airport environment created using Unreal Engine ™, the framework is demonstrated and validated by detecting and recognizing sequential gestures from a ground crew, who sends gesture commands to control the movement of an autonomous aircraft. A hybrid optimal controller is developed to combine the visual information obtained from the framework and prior information, such as airport map and reference lines, to control the aircraft to safely navigate to a terminal gate.

I. Nomenclature

$\mathbf{I}$	=	RGB image sequence
$\mathcal{A}$	=	marshalling signals set
c	=	discrete time step of an action change point
a	=	semantic label of a marshalling gesture signal
$\mathbf{v}$	=	human keypoint velocity
$\boldsymbol{\phi}$	=	Autoregressive process parameters
D	=	Mahalanobis distance
$\mathbf{h}$	=	Hu moments
w_r	=	time window size of the action recognition algorithm
w_d	=	time window size of the action detection algorithm
$\mathbf{s}$	=	hybrid system continuous state
μ	=	hybrid system discrete state
$\mathbf{u}$	=	hybrid system control input
d_a	=	user-specified alarm distance between the aircraft and the node
$\mathcal{G}$	=	simulated airport graph

[*]Ph.D. Student, Laboratory for Intelligent Systems and Controls (LISC), Sibley School of Mechanical and Aerospace Engineering.
[†]Master Student, Laboratory for Intelligent Systems and Controls (LISC), Sibley School of Mechanical and Aerospace Engineering.
[‡]Postdoctoral Associate, Laboratory for Intelligent Systems and Controls (LISC), Sibley School of Mechanical and Aerospace Engineering.
[§]Professor, Electrical and Computer Engineering Department.
[¶]Professor, Laboratory for Intelligent Systems and Controls (LISC), Sibley School of Mechanical and Aerospace Engineering.

v	=	aircraft speed
ψ	=	aircraft yaw angle
θ	=	aircraft steering angle
β	=	aircraft acceleration
l	=	distance between front and rear axles of the aircraft
v_d	=	desired aircraft speed

II. Introduction

Understanding a sequence of human actions and taking prompt response is crucial to many intelligent systems to complete complex tasks in human-centered environments, such as an assistive robot identifying the person-of-interest in the scene by recognizing a key action and starting following the target to execute any further visual command from them. Another example is advanced surveillance systems, which monitor a series of activities from the crowd to detect and respond to abnormal behaviors. Although action recognition algorithms are thought to play a crucial role in these applications, most of the existing approaches either focused on single-action videos or assumed a user could specify the time window *a priori* [1–5]. Recently, some deep learning algorithms [6–8] have been proposed to detect the starting and ending frames of the action of interest from an unsegmented raw video. However, they assume the entire video is observed by the network. Online action detection algorithms [9, 10] classify the present action from a streaming video with only past and current frames available. Their methods assume the video only contains one "key" action, while other events are background or irrelevant. This assumption limits the ability of the algorithms to comprehend a series of moves from the human. Besides, it is hard to transfer them to mobile robotic systems due to the high computational requirement of deep neural networks.

This paper develops a novel approach for detecting each action from an unsegmented ongoing image sequence consisting of multiple actions. The algorithm is composed of two stages. In the first stage, an online action change detection algorithm is developed to find the change point between two distinct actions. Unlike existing approaches that focus on temporally localizing a key action, this method provides an accurate starting frame of each action. In the second phase, the recognition module will only be applied if a new action change point is discovered. It alleviates the high computational cost of the conventional algorithms, which constantly recognizes video clips from every time window. The effectiveness of the algorithm is demonstrated by conducting numerical studies on a publicly available dataset [11]. Besides, the framework is integrated with a vision-guided autonomous taxiing system for the arrival simulation of an aircraft at the near-to-gate region. The airport environment is created by [12] using Unreal Engine ™[13], which is a high-fidelity physics-based simulation tool. In the synthetic experiment, the gesture commands from an airport ground crew are detected and recognized by the framework. A hybrid controller is developed to control the movement of an airplane based on the visual commands as well as tracking the feasible paths.

The paper is organized as follows. Section III formulates the two-stage action detection problem for vision-guided control of the aircraft ground movement. Section IV describes the background of the change detection method. Section V develops the two-stage action detection framework. Section VI presents a hybrid controller that switches the cruising mode based on the action command while keeping the aircraft tracking the feasible centerlines. The performance of the framework is demonstrated in Section VII, where a simulation is conducted for the arrival of an aircraft from the taxiway to a designated terminal gate.

III. Problem Formulation

This paper considers the problem of automatically recognizing distinct marshalling commands from a streaming image sequence for guiding the autonomous taxiing process of a near-to-gate aircraft. Let $\mathcal{A} = \{a_1 \ \dots \ a_n\}$ denote an unknown action set composed of standard marshalling gestures, where a_i represents the i^{th} action label. Denote the motion sequence consisting of image frames, $I_i \in \mathbb{R}^{h\times w}$, $(i = 1, \cdots, k)$ by the vector $\mathbf{I}_{1:k} = [I_1 \ \dots \ I_k]$. The goal of action recognition is to design a classification function F that assigns each frame of the motion sequence to its corresponding action label, i.e., $F(I_i) \in \mathcal{A}$. This is accomplished by a two-stage action detection framework as shown in Fig 1. Let c_i denote the time step of i^{th} change of action type in the sequence. In the first stage, the point is detected with a small time delay by a light-weight online action change detection approach $c_i = G(\mathbf{I}_{c_{i-1}:c_{i-1}+w_d})$, where w_d is the detection window size. Once c_i is discovered, the image sequence $\mathbf{I}_{c_i:c_i+w_r}$ will be processed by an effective action recognition method to produce a unique label $a_i = F(I_j)$, $j \in [c_i, c_i + w_r]$ for all the frames in the window, where w_r denotes the recognition window size. A feedback control system is designed to control the aircraft to safely navigate

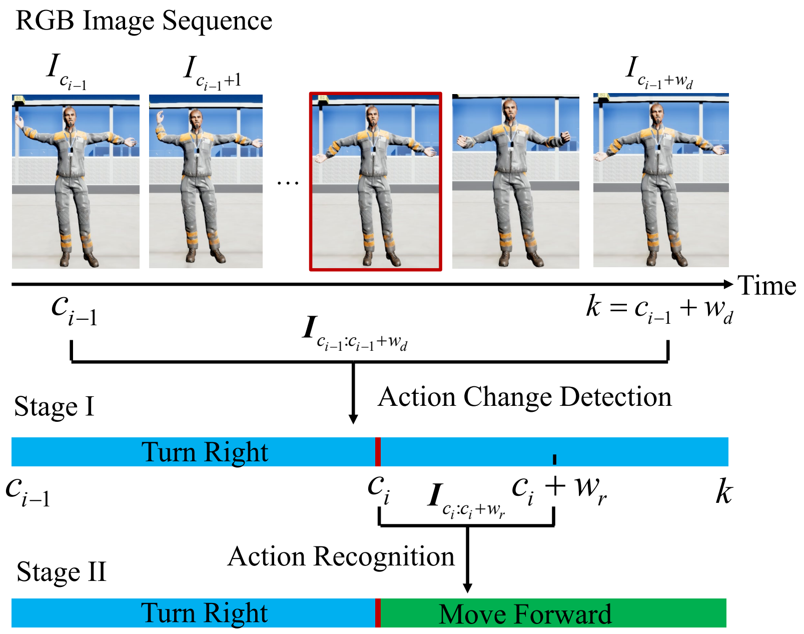

Fig. 1 The proposed two-stage framework. In the first stage, the change point is detected between two adjacent actions. The new action is recognized in the second phase.

through the airport environment based on visual guidance, as shown in Fig 2. Assuming the airport graph $\mathcal{G}$ that contains the information about the taxiways and terminal regions is available, a set of corresponding feasible cruising paths $\mathcal{H}$ can be generated for the aircraft to track. A hybrid controller is proposed to compute the control input and decide the cruising mode based on the recognized marshalling command, feasible paths, and state feedback. Section III.A formulates the gesture command sequence mathematically. The aircraft model and the airport model are discussed in Section III.B and III.C, respectively.

A. Ground Crew Marshalling Gesture Sequence

Let $a(k) \in \mathcal{A}$ denote the gesture label of the target ground crew $\mathcal{T}$ at time step k, where the action label set is defined as $\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5\} \triangleq$ {"Move Forward", "Turn Right", "Stop", "None", "Turn Left"}. As shown in Fig 3, the actions are standard marshalling commands for the near-to-gate navigation of the aircraft. During the simulation, it is assumed that $\mathcal{T}$ is always within the field of view (FoV) of a fixed RGB camera. The entire body of $\mathcal{T}$ can be projected to the camera frame, which is then processed by a real-time 2D pose estimation algorithm, such as [14], to

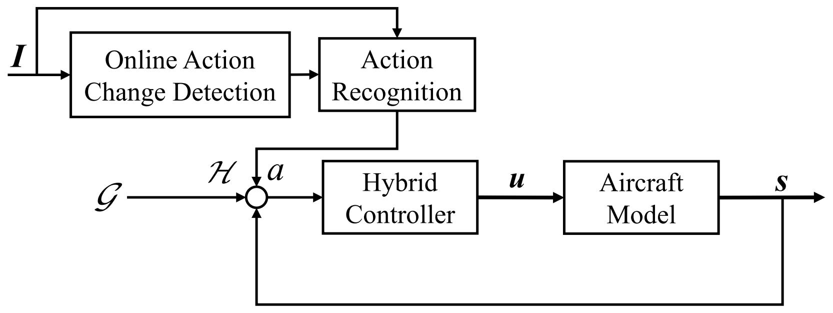

Fig. 2 Block diagram of the system. A set of feasible reference paths $\mathcal{H}$ is generated based on the airport graph $\mathcal{G}$. A sequence of gesture images I is processed online by the two-stage framework. A hybrid controller computes the control input u of the system based on the gesture label a, reference paths $\mathcal{H}$ and state feedback s.

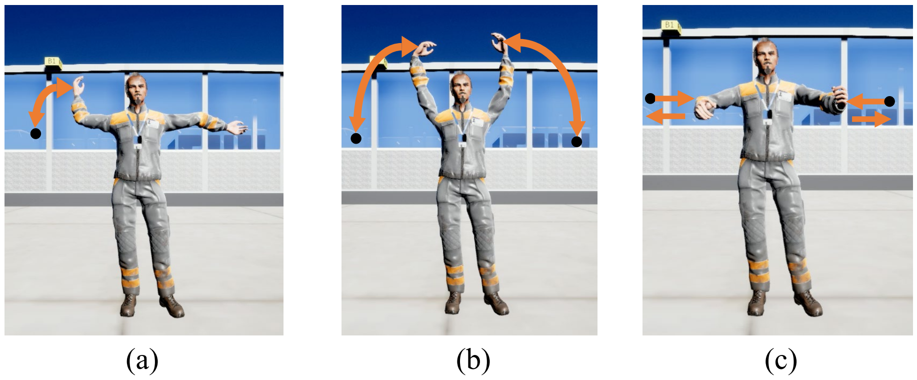
Fig. 3 Standard marshalling gesture signals (a) Turn right (b) Stop (c) Move Forward.

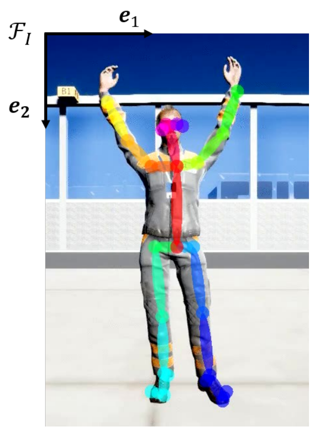

Fig. 4 Estimated keypoints of the ground crew using [14]. $\mathcal{F}_I$ represents the image coordinate system.

effectively represent human motions.

As shown in Fig 4, positions of the estimated keypoints are represented in a fixed image coordinate $\mathcal{F}_I$ with an ordered basis $\{\mathbf{e}_1, \mathbf{e}_2\}$. A velocity vector $\mathbf{v}_k$ of the keypoints at time step k is constructed to characterize motions,

$$\mathbf{v}_k = [\mathbf{v}_{k,x}^T \quad \mathbf{v}_{k,y}^T]^T \tag{1}$$

where $\mathbf{v}_k \in \mathbb{R}^{2d\times 1}$, $\mathbf{v}_{k,x} \in \mathbb{R}^{d\times 1}$, $\mathbf{v}_{k,y} \in \mathbb{R}^{d\times 1}$ are one-dimensional vectors, d is the number of keypoints. As an example, $\mathbf{v}_{k,x}$ can be written as,

$$\mathbf{v}_{k,x} = [v_{k,x_1} \;\; v_{k,x_2} \;\; \ldots \;\; v_{k,x_d}]^T \tag{2}$$

where v_{k,x_i} is the velocity of ith keypoint in $\mathbf{e}_1$ direction. The direction of The velocity vector is sorted in the same way as the keypoints order from [14]. Then, the vectors within the same time window are combined to formulate a multi-dimensional time sequence, i.e., $\mathbf{V} = [\mathbf{v}_{k-w_d}, \; \mathbf{v}_{k-w_d+1}, \; \ldots \; , \mathbf{v}_k] \in \mathbb{R}^{2d\times w_d}$, which is processed by the action change detection algorithm to produce the change point $c_i, i \in [1, n]$.

B. Aircraft Motion Model

This paper adopts the kinematic model used in [12] for airplane ground movement manipulation. Let $[x_k, y_k]$ denote the coordinates of the rear-axle center of the aircraft in inertial frame at time step k. Let ψ_k and v_k denote the aircraft yaw angle and speed respectively. Then, the aircraft ground state can be defined as $\mathbf{s}_k = [x_k, y_k, \psi_k, v_k]^T$. Denote the steering angle and forward acceleration as θ and β respectively. The control input can be expressed in the form of $\mathbf{u}_k = [\theta_k, \beta_k]^T$. The kinematic equation of the aircraft is shown as,

$$\mathbf{s}_{k+1} = \mathbf{f}(\mathbf{s}_k, \mathbf{u}_k) \tag{3}$$

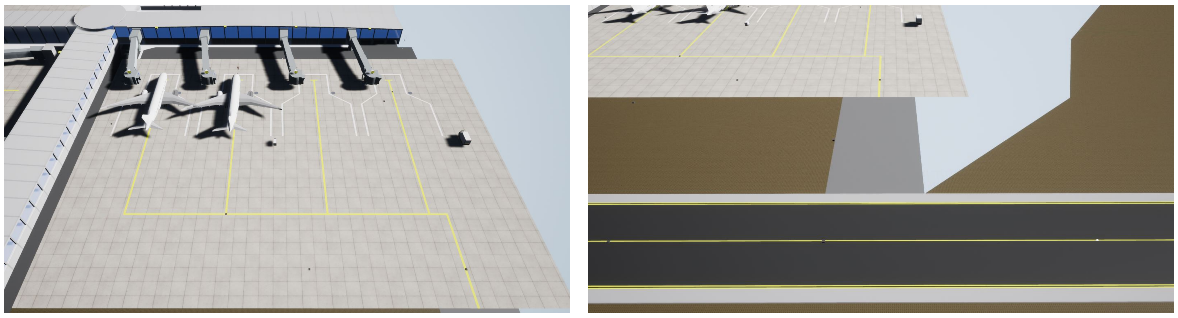

Fig. 5 Simulated airport environment generated by [12] using Unreal Engine ™. The terminal region (a) contains four gates and is connected by a taxiway (b) to the rest part of the airport.

where the difference function is given by a non-holonomic simple car model,

$$\mathbf{f}(\mathbf{s}_k, \mathbf{u}_k) = \begin{bmatrix} x_k \\ y_k \\ \psi_k \\ v_k \end{bmatrix} + \begin{bmatrix} v_k \cos\psi_k \\ v_k \sin\psi_k \\ \frac{v_k}{l} \tan\theta_k \\ \beta_k \end{bmatrix} \Delta t \tag{4}$$

where l is the distance between the front and rear axles of the aircraft, and Δt is the sampling time interval.

C. Airport Model

The airport environment is generated by [12] using Unreal Engine™, which is a photo-realistic physics based simulation tool. As shown in figure 5, the region of interest is characterized by a terminal with four gates (a) and a taxiway (b). Let $\mathcal{W} \in \mathbb{R}^2$ denote the workspace of the 2D ground plane of the region. The topological graph of the airport can be defined as $\mathcal{G} = (\mathcal{E}, \mathcal{V})$, where $\mathcal{E}$ is a set of undirected arcs that connect the nodes. A node $v \in \mathcal{V}$ is defined as a tuple $v = (\alpha, \mathbf{p}, \psi)$, where $\alpha \in \mathcal{L} = \{$"terminal", "connecting arcs", "taxiway"$\}$ is the label of the node, $\mathbf{p} \in \mathcal{W}$ is the position of the node, and $\psi \in (-\frac{\pi}{2}, \frac{\pi}{2})$ is the desired yaw angle of the aircraft at the node. To control the aircraft to cruise safely in the near-to-gate region, reference lines between any pair of connected nodes are generated for centerline following. Let $\mathcal{H} = \{h_{ij}(\mathbf{q}) = 0 | i, j \in [1, n], i \neq j\}$ denote a set that contains all the reference lines of the simulated airport, where $h_{ij}(\mathbf{q}) = 0$ is a line that connects the node v_i and v_j. The waypoints on the line is represented by $\mathbf{q} = [x_r, y_r] \in \mathcal{W}$.

Figure 5 illustrates the graph generated for the simulated airport. Here, v_s denotes the starting point of the aircraft. It is followed by two taxiway nodes v_1 and v_2, which are then connected to transition nodes v_3 and v_4 via dashed curves. Through transition node v_3, the airplane can cruise from taxiway to intermediate terminal node v_5 or v_6. Similarly, the aircraft can reach v_7 or v_8 via transition node v_4. The gate nodes include v_9, v_{10}, v_{11} and v_{12}, which are connected to v_5, v_6, v_7 and v_8 respectively. The ground crew will send gesture command to the airplane when it approaches or arrives at a node to direct it to the next node.

IV. Background on Multiple Change Point Analysis Method

Our online action change detection algorithm is based on Multiple change point analysis (MCPA) [15], which outperforms other binary change point detection methods [16] in terms of computation complexity, detection performance, and robustness. To date, the algorithm has been applied to the problem of detecting change points in one-dimensional time series data. The method assumes the data is composed of several segments, each of which is generated by an autoregressive (AR) process. Specifically, consider one-dimensional time sequence $\mathbf{Y} = [Y_1 \ \ldots \ Y_n]$, which is generated by an unknown number of distinct segment-wise AR process. MCPA method slides a time window L to split the data into m equal length segments. For an arbitrary data point Y_i in jth segment, where $j \in [1, m]$, it is generated by an AR

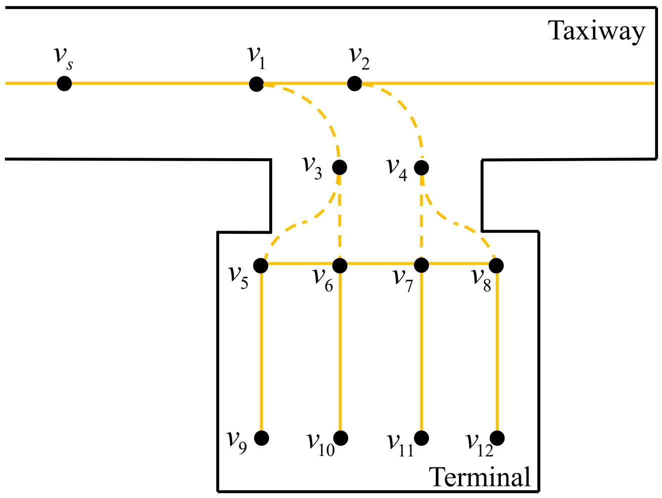

Fig. 6 A topological graph illustration of the simulated airport map. The black dots represent the nodes and continuous yellow lines are the arcs that connect them. The dashed lines connect taxiway with the terminal region.

process:

$$Y_i = \epsilon_{i,j} + [1 \;\; Y_{i-1} \;\; \dots \;\; Y_{i-p}]\boldsymbol{\phi}_j \tag{5}$$

where $\epsilon_{i,j}$ are i.i.d. Gaussian noise terms. The parameters of the AR filter are expressed as, $\boldsymbol{\phi}_j = [c \; \alpha_1 \; \dots \; \alpha_p]^T$, where p is the order of the AR process. $\boldsymbol{\phi}_j$ is estimated by minimizing accumulated prediction error for each point in jth segment:

$$\underset{\boldsymbol{\phi}_j}{\operatorname{argmin}} \sum_{i \in \mathcal{B}_j} (Y_i - [1 \;\; Y_{i-1} \;\; \dots \;\; Y_{i-p}]\boldsymbol{\phi}_j)^2 \tag{6}$$

where $\mathcal{B}_j$ is the index set of jth segment. Then, the estimation is repeated to obtain the vectors of AR parameters for all of the segments: $[\boldsymbol{\phi}_1^T \; \dots \; \boldsymbol{\phi}_m^T]$. Next, the algorithm clusters similar AR parameter vectors together by selecting the best number of groups and the range for each group. Assuming the data contains $M + 1$ groups, the optimal range for all groups are estimated by minimizing the accumulated within-group variance of AR parameter vectors,

$$\underset{[S_1,\dots,S_M]}{\operatorname{argmin}} \sum_{q=1}^{M+1} \lambda(\boldsymbol{\phi}_{S_{q-1}+1}, \dots, \boldsymbol{\phi}_{S_q}) \tag{7}$$

$$\lambda(\boldsymbol{\phi}_{S_{q-1}+1}, \dots, \boldsymbol{\phi}_{S_q}) = \sum_{j=S_{q-1}+1}^{S_q} |\boldsymbol{\phi}_j - \overline{\boldsymbol{\phi}}_q|^2 \tag{8}$$

where the start points of each group are indexed by $[S_0, S_1, \dots, S_M]$ respectively ($S_{M+1} = m$), the variable $\overline{\boldsymbol{\phi}}_q = \sum_{j=S_{q-1}+1}^{S_q} \boldsymbol{\phi}_j / (S_q - S_{q-1})$ is the mean of the parameter vector group. Then, MCPA algorithm iterates the minimization process by changing the value of M from one to a user-defined limit. The optimal value of M correspond to minimum accumulated within-group variance. Once the clustering is achieved, change points are identified at the boundary segments of neighboring groups. These regions are given a score of one. Naturally, the algorithm tends to select a large M to minimize within-group variance. To choose optimal number of change points, MCPA method uses BIC-like criterion to design a penalty function $\beta_M = M\text{var}(\boldsymbol{\phi})\log\log n$, where M is the number of change points. Finally, a multi-window process is developed, where different window sizes are selected. Scores from each window size are accumulated together to formulate the final scores of the detection. A region associated with a high score is likely to contain a change point. More details of the MCPA algorithm can be found in algorithm 1.

Algorithm 1 Multiple Change Point Analysis Method (MCPA)

Input: time series $\mathbf{Y} = [Y_1 \ Y_2 \ \dots \ Y_n]$, window sizes $[w_{\min} \ \dots \ w_{\max}]$, order p for AR model
Output: score $\mathbf{s}$, optimal loss e^*_M, change point c.

```
1:  for $w = [w_{\min} \ \dots \ w_{\max}]$ do
2:      Define maximum number of change points $M_w$
3:      Calculate $\{\boldsymbol{\phi_j}, j = 1, \dots, n/w\}$
4:      for $M = 1, \dots, M_w$ do
5:          Define $S_0 = 0, S_{M+1} = n/w$
6:          Define Loss function $\lambda(\boldsymbol{\phi}_{S_{q-1}+1}, \dots, \boldsymbol{\phi}_{S_q}) = \sum_{j=S_{q-1}+1}^{L_q} |\boldsymbol{\phi}_j - \overline{\boldsymbol{\phi}}|^2$
7:          Define penalty terms $\beta_M$
8:          min $e_M = \sum_{q=1}^{M+1} \lambda(\boldsymbol{\phi}_{S_{q-1}+1}, \dots, \boldsymbol{\phi}_{S_q}) + \beta_M$
9:          Record optimal $e_M$ and ranges $[S_1, \dots, S_M]$
10:     end for
11:     Find the optimal $M^*$, and corresponding loss $e^*_M$, segment length $[S^*_1, \dots, S^*_M]$
12:     Accumulate score $\mathbf{s} = \mathbf{s} + \mathbf{1}$ at the boundary segments between two optimal groups.
13: end for
14: Find the change point $c$ using $\mathbf{s}$. The details are described in [15].
```

V. Online Action Change Detection

The two-stage action detection framework is described in detail in this section. Given an untrimmed streaming sequence of the target gestures, an online action change detection algorithm is developed to locate the change point accurately with a small time delay. The method is described in Section V.A. In the second phase, an action recognition module will be applied to the streaming image sequence with the change point information to obtain the action label. The classification method is discussed in Section V.B.

A. Online Action Change Point Detection

A multi-variate online action change detection algorithm is introduced in this section. The autoregression is extended to vector autoregression (VAR) for handling the high-dimensional skeleton data. Compared to the full VAR model, block diagonal VAR model is found to give the best trade-off between detection accuracy and computation time. Given an arbitrary window of joint velocity sequence $\mathbf{V} = [\mathbf{v}_1 \ \mathbf{v}_2 \ \dots \ \mathbf{v}_{w_d}] \in \mathbb{R}^{2d \times w_d}$, it is first divided into m number of non-overlapping segments by a sliding window with length L. Without loss of generality, consider a velocity vector $\mathbf{v}_i$ in j^{th}, $j \in [1, m]$ segment, its VAR expression can be written in the following form,

$$\mathbf{v}_i = \begin{bmatrix} v_{i,x_1} \\ v_{i,x_2} \\ \vdots \\ \vdots \\ v_{i,y_d} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\phi}^T_{j,x_1} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\phi}^T_{j,x_2} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \mathbf{0} & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{\phi}^T_{j,y_d} \end{bmatrix} \begin{bmatrix} \mathbf{v}_{i-p,x_1} \\ \mathbf{v}_{i-p,x_2} \\ \vdots \\ \vdots \\ \mathbf{v}_{i-p,y_d} \end{bmatrix} + \boldsymbol{\epsilon_j} \tag{9}$$

where the velocity vector on right-hand-side is defined as $\mathbf{v}_{i-p,x_i} = [1 \ v_{i-1,x_i} \ \dots \ v_{i-p,x_i}]^T$, p is the order of the VAR process. The VAR parameter vector is denoted by $\boldsymbol{\phi}_{j,x_i} = [\tau_{j,x_i} \ \alpha_{1,j,x_i} \ \dots \ \alpha_{p,j,x_i}]^T$. Besides, $\boldsymbol{\epsilon}_j$ is the i.i.d. Gaussian noise vector for j^{th} segment. The parameters of the equation can be estimated via least square method, let $\boldsymbol{\Phi_j} \in \mathbb{R}^{d(p+1) \times d(p+1)}$ denote the entire block diagonal VAR parameter matrix. Concatenating all the velocity vectors to a long vector $\mathbf{v}_{i-p} = [\mathbf{v}^T_{i-p,x_1} \ \dots \ \mathbf{v}^T_{i-p,y_d}]^T \in \mathbb{R}^{d(p+1) \times 1}$. Then, the least square estimation can be written as below,

$$\underset{\boldsymbol{\Phi_j}}{\operatorname{argmin}} \sum_{i \in \mathcal{B}_j} ||\mathbf{v}_i - \boldsymbol{\Phi_j} \mathbf{v}_{i-p}|| \tag{10}$$

where $\mathcal{B}_j$ is the index set of jth segment. In a similar fashion, the VAR parameter vectors for other segments can be estimated. Let $\boldsymbol{\Phi} \in \mathbb{R}^{d(p+1) \times md(p+1)}$ denote the matrix of combination of all the VAR parameters from different segments. The original sequence $\mathbf{V}$ is now transformed to $\boldsymbol{\Phi}$, which is in a reduced-dimension parameter space.

Different from [15] that assumes all the time series data is available and observed, an online scheme is designed to process the streaming data. First, assume an initial velocity sequence $\mathbf{V}_1 = [\mathbf{v}_1 \ \ldots \ \mathbf{v}_{w_d}]$, which is transformed to VAR parameters $\mathbf{\Phi}_1$. It is assumed that there are zero or only one change point in the window. Then, algorithm 1 is called to obtain the optimal loss e^*. It is worth noting that the penalty function β can be dropped from the loss since it is not helpful for our one-change point assumption. Comparing e^* to the variance e of $\mathbf{\Phi}_1$. If $e^* < eh$, the change point is considered existing. Parameter h needs to be manually tuned for different applications. If a change point is detected, the next time window will start from that point and continue receiving data until the length w_d is reached. Otherwise, the first $L_{\max}$ number of vectors in the window will be substituted by the incoming $L_{\max}$ number of data. Algorithm 2 describes the online change detection process in detail.

Algorithm 2 Online Action Change Detection

```
Input: Current velocity sequence $\mathbf{V}_m$, current velocity vector $\mathbf{v}_k$, and parameter $h$, $w_d$, $L_{\max}$
Output: Change point $c_i$, updated velocity sequence $\tilde{\mathbf{V}}_m$
 1: function $c_i, \tilde{\mathbf{V}}_m$ = ChangeDetection($\mathbf{V}_m, \mathbf{v}_k, w_d, h, L_{\max}$)
 2:     if length($\mathbf{V}_m$) $< w_d$ then
 3:         $c_i \leftarrow \emptyset$
 4:         $\tilde{\mathbf{V}}_m$ = SlidingWindow($\mathbf{V}_m, \mathbf{v}_k, c_i, 0, L_{\max}$)
 5:     else
 6:         $\tilde{e}, e, \hat{c}_i$ = MCPA($\mathbf{V}_m$)
 7:         if $\tilde{e} < eh$ then
 8:             $c_i = \hat{c}_i$
 9:         else
10:             $c_i \leftarrow \emptyset$
11:         end if
12:         $\tilde{\mathbf{V}}_m$ =SlidingWindow($\mathbf{V}_m, \mathbf{v}_k, c_i, 1, L_{\max}$)
13:     end if
14:     return $c_i, \tilde{\mathbf{V}}_m$
15: end function
Input: Current velocity vector $\mathbf{v}_k$, change point $c_i$, flag and $L_{\max}$
Output: Updated velocity sequence $\tilde{\mathbf{V}}_m$
 1: function $\tilde{\mathbf{V}}_m$ = SlidingWindow($\mathbf{V}_m, \mathbf{v}_k, c_i$, flag, $L_{\max}$)
 2:     if flag == 0 then
 3:         $\tilde{\mathbf{V}}_m \leftarrow$ append $\mathbf{v}_k$ to the end of $\mathbf{V}_m$
 4:         return $\tilde{\mathbf{V}}_m$
 5:     else
 6:         if $c_i \neq \emptyset$ then
 7:             $\tilde{\mathbf{V}}_m \leftarrow$ remove velocities before time step $c_i$, append $\mathbf{v}_k$ to the end of $\mathbf{V}_m$
 8:             return $\tilde{\mathbf{V}}_m$
 9:         else
10:             $\tilde{\mathbf{V}}_m \leftarrow$ remove velocities before time step $L_{\max}$, append $\mathbf{v}_k$ to the end of $\mathbf{V}_m$
11:             return $\tilde{\mathbf{V}}_m$
12:         end if
13:     end if
14: end function
```

B. Action Recognition

Assuming the last change point before current time step k is c_i. If $k - c_i > w_r$, where w_r is the length of the recognition window, the action recognition function F will be applied on the video clip $\mathbf{I}_{c_i:c_i+w_r} = [I_{c_i}, \ I_{c_i+1}, \ \ldots, \ I_{c_i+w_r}]$ to identify the semantic labels,

$$a(j) = F(\mathbf{I}_{c_i:c_i+w_r}), \ j = c_i, \ldots, k \tag{11}$$

where the action labels are the same from step c_i to k. Here, an action recognition algorithm [17] is implemented. The method employs motion energy image (MEI) and motion history image (MHI) to represent motions in an image sequence or a video clip. MEI is a cumulative binary motion image, which indicates where the pixels are moving in the image. MHI is a scalar-valued image where more recently moving pixels are brighter. Assume $k = c_i + w_r$, the MEI and MHI are defined as,

$$E_k = \sum_{i=0}^{w_r-1} I_{k-i} - I_{k-i-1} \tag{12}$$

$$H_k = \begin{bmatrix} H_{0,0,k} & H_{1,0,k} & \cdots & H_{w,0,k} \\ H_{0,1,k} & H_{1,1,k} & \cdots & H_{w,1,k} \\ \vdots & \vdots & \vdots & \vdots \\ H_{0,h,k} & H_{1,h,k} & \cdots & H_{w,h,k} \end{bmatrix} \tag{13}$$

where each entry in H_k is defined as,

$$H_{\xi,\eta,k} = \begin{cases} w_r & \text{if } |I_{\xi,\eta,k} - I_{\xi,\eta,k-1}| > d_c \\ \max(0, H_{\xi,\eta,k-1} - 1) & \text{otherwise} \end{cases} \tag{14}$$

where $|I_{\xi,\eta,k} - I_{\xi,\eta,k-1}|$ is the pixel intensity difference at location (ξ, η), d_c is a pre-defined threshold.

H_k and E_k can effectively represent the shape and temporal history of human motion. The first seven Hu moments are used as a feature descriptor of H_k and E_k. They are invariant to scale, translate and rotate. Mahalanobis distance $D_{i,j}$ is used to calculate the similarity between the hu moments $\mathbf{h}_{c_i:c_i+w_r}$ of the testing sample and that of the training action,

$$D_{i,j} = \sqrt{(\mathbf{h}_{c_i:c_i+w_r} - \overline{\mathbf{h}}_j)^T \mathbf{K}_j^{-1} (\mathbf{h}_{c_i:c_i+w_r} - \overline{\mathbf{h}}_j)} \tag{15}$$

where $\mathbf{K}_j \in \mathbb{R}^{7\times7}$ and $\overline{\mathbf{h}}_j \in \mathbb{R}^{7\times1}$ are the covariance and mean of the Hu moments of the a_j respectively, $\mathbf{h}_{c_i:c_i+w_r} \in \mathbb{R}^{7\times1}$ is the Hu moment vector of the testing sample collected on $\mathbf{I}_{c_i:c_i+w_r}$. The metric will produce a smaller value for actions with similar shape and trajectory history.

VI. Hybrid Control of the Near-to-Gate Aircraft

A hybrid control system is designed with inspiration from [12] to control the aircraft to reach a terminal gate designated by the ground crew. The controller in [12] assumes the Air Traffic Control (ATC) commands are received in advance for path planning. Different from that, the control system in this paper processes the incoming marshalling signals to coordinate between the discrete modes, including "Cruising Forward", "Turning", "Holding" and "Stop" , as well as tracking the reference line. The "Cruising Forward" and "Turning" modes are solely determined by the action recognition result on the ground crew gesture command. The "Holding" mode is activated when the plane approaches a transit node without receiving any new instructions. The "Stop" mode is triggered when the aircraft arrives at the terminal gate and receives the corresponding gesture command.

A. Continuous State Control Law

"Cruising Forward" Mode. The airplane is controlled to move forward with a pre-defined desired speed v_d along the centerline path generated between the current node and the following node. If there are multiple branches, the angle of the vector formed by the current position of the aircraft and each potential node will be calculated. The branch with the correct angle that aligns with the "Forward" command will be selected. This control scheme is triggered when the airplane is instructed to "Cruising Forward". The waypoints $[\mathbf{q}_k{}^T, \ldots, \mathbf{q}_{k_f}{}^T]^T = [x_{r,k}, y_{r,k}, \ldots, x_{r,k_f}, y_{r,k_f}]^T$ and reference yaw angles $[\psi_{r,k}, \ldots, \psi_{r,k_f}]^T$ are assumed to be known in the inertial frame. The reference state at time step j is then defined as $\mathbf{s}_{r,j} = [\mathbf{q}_k{}^T,\ \psi_{r,k},\ v_d]^T$. An optimal control problem is constructed for this mode. The control objective function is defined as,

$$J_{\mu_1} = ||\mathbf{s}_{k_f} - \mathbf{s}_{r,k_f}||_2^2 + \sum_{j=k}^{k_f} ||\mathbf{s}_j - \mathbf{s}_{r,j}||_2^2 + ||\mathbf{u}_{\mu_1,j}||_2^2 \tag{16}$$

The deviation from the centerline and control usage are penalized. Then, a nonlinear optimization problem can be formulated as below,

$$\begin{aligned} \min_{\mathbf{x}_1} \quad & J_{\mu_1} \\ \text{subject to} \quad & \mathbf{s}_{j+1} = f(\mathbf{s}_j, \mathbf{u}_{\mu_1,j}), j = k, \ldots, k_f - 1 \\ & \mathbf{s}_j \in \mathcal{S}, j = k, \ldots, k_f \\ & \mathbf{u}_{\mu_1,j} \in \mathcal{U}, j = k, \ldots, k_f \\ & \mathbf{s}_k = \mathbf{s}_0 \end{aligned} \tag{17}$$

where $\mathbf{s}_0$ is the initial condition of the state, $\mathcal{S}$ and $\mathcal{U}$ are a set of admissible states and control inputs of the aircraft respectively. All the states and control inputs in the current time window are organized as the following vector and optimized together,

$$\mathbf{x}_1 = [\mathbf{s}_k^T, \ldots, \mathbf{s}_{k_f}^T, \mathbf{u}_{\mu_1,k}^T, \ldots, \mathbf{u}_{\mu_1,k_f}^T]^T \tag{18}$$

"Turning" Mode. The airplane is controlled to turn toward or turn away from the terminal gate while following the centerline path and maintaining a pre-defined desired speed v_d. When there are multiple branches of the centerline, the angle of the vector that connects the current position of the aircraft and the destination node of each branch will be estimated. The airplane will follow the branch whose estimated angle aligns with the gesture command. The control objective function is the same to the "Cruising Forward" mode and is defined as,

$$J_{\mu_2} = ||\mathbf{s}_{k_f} - \mathbf{s}_{r,k_f}||_2^2 + \sum_{j=k}^{k_f} ||\mathbf{s}_j - \mathbf{s}_{r,j}||_2^2 + ||\mathbf{u}_{\mu_2,j}||_2^2 \tag{19}$$

The nonlinear minimization problem is constructed as,

$$\begin{aligned} \min_{\mathbf{x}_2} \quad & J_{\mu_2} \\ \text{subject to} \quad & \mathbf{s}_{j+1} = f(\mathbf{s}_j, \mathbf{u}_{\mu_2,j}), j = k, \ldots, k_f - 1 \\ & \mathbf{s}_j \in \mathcal{S}, j = k, \ldots, k_f \\ & \mathbf{u}_{\mu_2,j} \in \mathcal{U}, j = k, \ldots, k_f \\ & \mathbf{s}_k = \mathbf{s}_0 \end{aligned} \tag{20}$$

The optimization variables are,

$$\mathbf{x}_2 = [\mathbf{s}_k^T, \ldots, \mathbf{s}_{k_f}^T, \mathbf{u}_{\mu_2,k}^T, \ldots, \mathbf{u}_{\mu_2,k_f}^T]^T \tag{21}$$

"Holding" Mode. When the airplane is within the alarm distance d_a of the next transit node and no further command is received, the airplane will be controlled to reach the node while decelerating to zero velocity when it arrives. The location of the nodes and centerline waypoints are assumed to be known in the inertial frame. The control objective function is defined as,

$$J_{\mu_3} = ||\mathbf{s}_{k_f} - [x_n, y_n, \psi_n, 0]^T||_2^2 + \sum_{j=k}^{k_f} ||\mathbf{s}_j - [x_{r,j}, y_{r,j}, \psi_{r,j}, 0]^T||_2^2 + ||\mathbf{u}_{\mu_3,j}||_2^2 \tag{22}$$

where $[x_n, y_n]^T$ is the location of the next node and θ_n is the desired yaw angle of the plane at the node. A nonlinear minimization problem is formulated below for finding the optimal control inputs,

$$\begin{aligned} \min_{\mathbf{x}_3} \quad & J_{\mu_3} \\ \text{subject to} \quad & \mathbf{s}_{j+1} = f(\mathbf{s}_j, \mathbf{u}_{\mu_3,j}, j = k, \ldots, k_f - 1 \\ & \mathbf{s}_j \in \mathcal{S}, j = k, \ldots, k_f \\ & \mathbf{u}_{\mu_3,j} \in \mathcal{U}, j = k, \ldots, k_f \\ & \mathbf{s}_k = \mathbf{s}_0 \end{aligned} \tag{23}$$

The optimization variables are defined as,

$$\mathbf{x}_3 = [\mathbf{s}_k^T, \ldots, \mathbf{s}_{k_f}^T, \mathbf{u}_{\mu_3,k}^T, \ldots, \mathbf{u}_{\mu_3,k_f}^T]^T \tag{24}$$

"Stop" Mode. When the airplane arrives at the designated terminal gate and receives the "Stop" command from the ground crew, the engine will shut down and remain at that state until instructed to start again. The control law for this mode is trivial,

$$\mathbf{u}_{\mu_4} = \mathbf{0} \tag{25}$$

B. Mode Switching Control Law

"Cruising Forward" Mode. The discrete controller decides when to switch the system state to another mode from "Cruising". If the aircraft receives "Turning" command from the ground crew, the aircraft will enter the corresponding mode. Besides, if the aircraft approaches a node without receiving any new gesture commands, the airplane will switch to "Holding" mode and start to decelerate to zero velocity. The plane will remain idle until the next instruction arrives.

$$\mu(k) = \begin{cases} \mu_2 & \text{if } a(k) = \text{"Turn Right" or "Turn Left"} \\ \mu_3 & \text{if } a(k) = \text{"None" and } ||[x_k, y_k]^T - [x_n, y_n]^T||_2^2 < d_a \\ \mu_1 & \text{otherwise} \end{cases} \tag{26}$$

"Turning" Mode. The discrete controller decides when to enter another system state from "Turning". If the ground crew instructs the plane to change from "Turning" to "Move Forward", the airplane will switch to "Cruising Forward" mode. Another situation is that when the plane approaches a node without receiving new commands, the airplane will automatically change to "Holding" state.

$$\mu(k) = \begin{cases} \mu_1 & \text{if } a(k) = \text{"Move Forward"} \\ \mu_3 & \text{if } a(k) = \text{"None" and } ||[x_k, y_k]^T - [x_n, y_n]^T||_2^2 < d_a \\ \mu_2 & \text{otherwise} \end{cases} \tag{27}$$

"Holding" Mode. The airplane can reach any other system states in the "Holding" mode. The switching scheme is solely based on the gesture commands. The discrete control law is defined as,

$$\mu(k) = \begin{cases} \mu_1 & \text{if } a(k) = \text{"Move Forward"} \\ \mu_2 & \text{if } a(k) = \text{"Turn Right" or "Turn Left"} \\ \mu_4 & \text{if } a(k) = \text{"Stop"} \\ \mu_3 & \text{otherwise} \end{cases} \tag{28}$$

"Stop" Mode. The airplane will shut down and remain in the "Stop" mode.

$$\mu(k) = \mu_4 \tag{29}$$

VII. Results

The performance of the two-stage action detection is evaluated on a publicly available real-world gesture dataset [11]. The robustness of the detection surpasses other action segmentation methods significantly using standard metrics. The action classification result is also described. Then, the automatic cruising of the aircraft from the taxiway to the designated gate is simulated. The results demonstrate the effectiveness of the vision-guided hybrid control system.

A. Two-Stage Online Action Detection Performance

The algorithm is tested on UTD Multimodal Human Action Dataset (UTD-MHAD) [11], which consists of 20 different human gestures, including daily actions and sports moves. Many of them are very similar, for instance, arms cross and arms curl, baseball swing and tennis swing. Therefore, it is an appropriate as well as a challenging dataset for testing the robustness and accuracy of the two-stage online action detection (OACD) algorithm. The dataset was collected by a fixed Microsoft Kinect sensor and a wearable inertial sensor in the indoor environment. In the simulation, gestures are randomly selected and concatenated together to form a continuous motion sequence. OpenPose [14] is used to extract keypoint velocity vectors from the video stream, which is then processed by the change detection algorithm frame by frame. The change points will be reported once it is discovered in the current sliding window. The parameters of the method are manually selected as $w_d = 200$, $\mathbf{L} = [40, 30, 10]$, $p = 1$ and $h = 0.1$.

The accuracy and robustness of the algorithm are evaluated numerically using the standard Precision and False Negative Rate (FNR) metric. Let n_{TP}, n_{FP}, n_{FN} respectively denote the number of correctly detected change points, the number of falsely detected change points, and the number of mistakenly undetected change points. The metrics can be written as below,

$$\text{Precision} = \frac{n_{\text{TP}}}{n_{\text{TP}} + n_{\text{FP}}} \tag{30}$$

$$\text{FNR} = \frac{n_{\text{FN}}}{n_{\text{TP}} + n_{\text{FN}}} \tag{31}$$

In the simulation, 20 randomly generated motion sequences are tested. Table 1 summarizes the Precision and FNR results of the OACD algorithm and compares them with other action segmentation methods. The low FNR indicates that the algorithms can robustly avoid miss detection. Despite that OACD is an online method, it has a precision that is close to that of other offline algorithms. Besides, the average runtime of the algorithm on one sliding window is only $0.012s$ with MATLAB implementation on an AMD Ryzen 3700X windows computer, which is suitable for real-time control problems.

Table 1
Online Action Change Detection Performance.

Setting	Methods	Precision	FNR
Offline	ACA [18]	0.88	0.35
Offline	HACA [19]	0.85	0.31
Offline	SC [20]	0.67	0.54
Online	OACD	0.77	**0.026**

The window size of the action recognition algorithm [17] is set to $w_r = 30$. The module is applied once a new change point is detected. The method is tested on 10 random sequences generated by the following actions $\mathcal{A} = \{$"Waving", "Boxing", "Swiping", "Waving", "Push", "Swing"$\}$. The results are quantified by the Precision metric, where a perfect accuracy is achieved. As a comparison, the method proposed in [17] for finding the correct time window by minimizing the Mahalanobis distance performs poorly on the continuous action sequences. It only achieves a value of 0.4 using the same metric on the same dataset.

B. Aircraft Arrival Simulation

The arrival of an aircraft from the taxiway to a designated terminal gate is simulated. A commercial airplane model is used with front-rear axle distance $l = 15m$, acceleration $\beta = [-10m/s^2, 10m/s^2]$, and yaw angle $\psi = [-\frac{\pi}{2}, \frac{\pi}{2}]$. The maximum speed of the aircraft is set to $10m/s$. Besides, the sampling interval is $\Delta t = 0.1s$ and the planning horizon is $k_f - k = 10$. At the beginning, the simulated aircraft idles at node v_s with zero velocity, as shown in Fig 7(a). Upon receiving the "Move Forward" command from the ground crew, it starts to accelerate to the desired velocity while following the centerline. Two scenarios are considered when the airplane approaches v_1. In the first case, the ground crew will send "Turn Right" command before the aircraft reaches the alarm distance d_a with respect to v_1. For the second simulation, the crew sends the command after the airplane is within the alarm distance d_a of v_1. The plane will first enter "Holding" mode then switch to "Turning" due to the command delay. Fig 7(b) shows the pose and position of the airplane in "Turning" mode. Next, the ground crew will send "Move Forward" command when the aircraft approaches node v_3. As shown in Fig 7(c), the airplane will be controlled to move forward since it receives the command before it is too close to v_3. Similarly, the crew member will send "Move Forward" again when the plane approaches node v_6 to direct it to the terminal gate v_{10}. The airplane will automatically decelerate to safely stop at the gate region. The ground crew will send "Stop" command to let the engine shut down. Fig 7(d) displays the final position of the airplane. In the simulation, the nodes are connected in below order,

$$v_s \rightarrow v_1 \rightarrow v_3 \rightarrow v_6 \rightarrow v_{10}$$

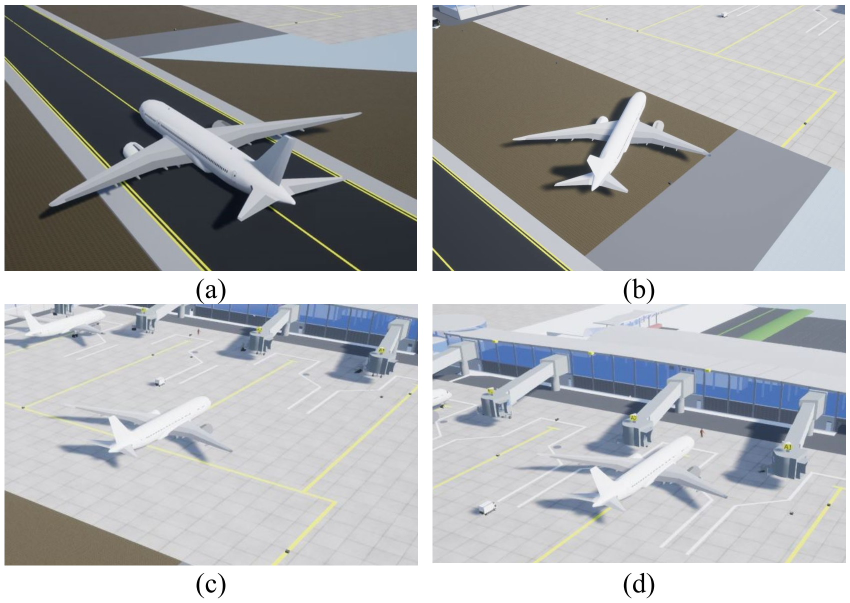

Fig. 7 Snapshots of the Arrival simulation. (a) The aircraft starts taxiing. (b) The airplane turns toward the designated terminal gate. (c) The aircraft approaches and gate. (d) The aircraft arrives and the engine shuts down.

Marshalling gestures are concatenated together to form the desired action sequence as described above. The time span of each motion segment is designed based on the approximate time that the airplane will spend on each mode, given the constraints. The sequence is ordered as below,

$$a_1 \to a_4 \to a_2 \to a_4 \to a_1 \to a_4 \to a_1 \to a_4 \to a_3$$

where the duration of each gesture is set to $6s, 9s, 9s, 6s, 10s, 9s, 10s, 12s$ and $11s$ respectively. As shown in Fig 8, the OACD algorithm can accurately detect all the change points with the largest error equaling to $1s$ at c_6. Once a new action segment is detected, it will be classified by the action recognition algorithm. Define the normalized M-distance for the testing sample $\mathbf{I}_{c_i:c_i+w_r}$ as,

$$\tilde{D}_{i,j} = \frac{D_{i,j}}{\max\{D_{i,j} \mid j \in [1,5]\}} \tag{32}$$

where the distance is divided by the largest distance among the testing sample and any of the training action $a_j \in \mathcal{A}$. Recall that the similarity between two actions is quantified by the proximity of their M-distance; a small distance means that the two samples possibly belong to the same action. Therefore, the smallest normalized distance $\tilde{D}_i^* = \min\{\tilde{D}_{i,j} \mid j \in [1,5]\}$ for each gesture segment is shown in Fig 8. All the actions are correctly recognized and most of them are classified with a high confidence. The time delay of the recognition for each segment is $\tau = [\tau_1, \ldots, \tau_8] = [2.90s, 1.23s, 0.03s, 0.70s, 0.70s, 0.37s, 2.70s, 1.33s]$.

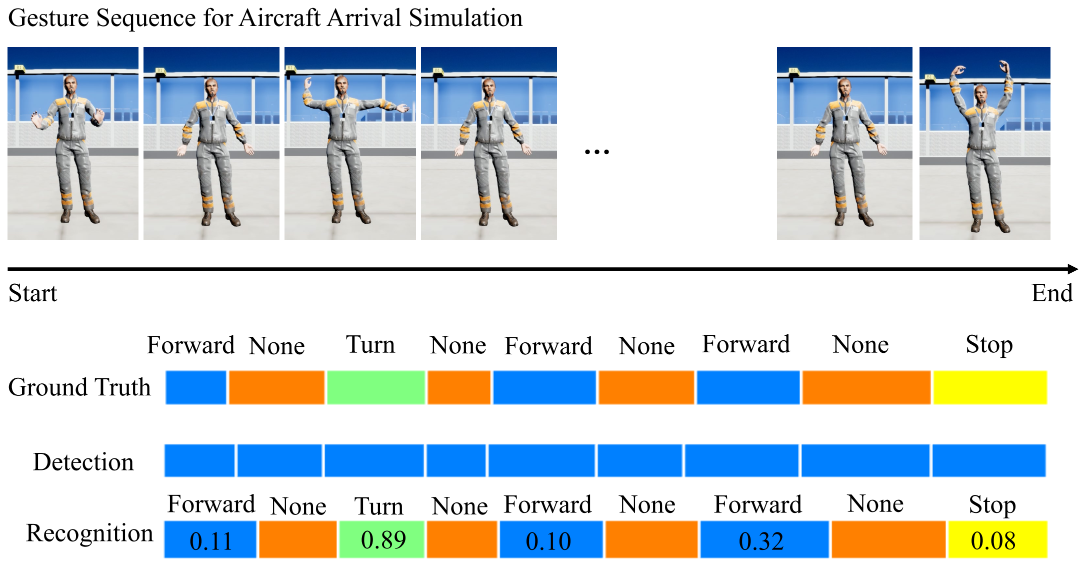

Fig. 8 The detection and recognition results on gesture sequence for arrival simulation. The two-stage action detection algorithm can accurately detect and recognize the actions with reasonable time delays. The values of $\tilde{D}_i^*$ are displayed over each action segment.

The resulting gesture label sequence $[a(1) \ \dots \ a(9)]$ is used by the hybrid controller to decide the cruising mode of the aircraft. Aside from the above action sequence, another sequence is generated for the second scenario, where the duration of $a(2)$ is extended to $10s$ such that the aircraft will first reach the distance d_a before it receives the "Turn Right" command. Fig. 9(a) and 10(a) show the actual trajectory of the aircraft compared to the reference line for the two scenarios respectively. The corresponding control inputs and aircraft speed are plotted in Fig. 9(b)-(c) and Fig. 10(b)-(c) . It can be observed that the aircraft successfully switches between the different modes and follows the desired centerline to reach the designated gate. Note that the optimization and gesture recognition are performed online using MATLAB® to pre-compute all the aircraft states, which are then sent to Unreal Engine ™ for simulation.

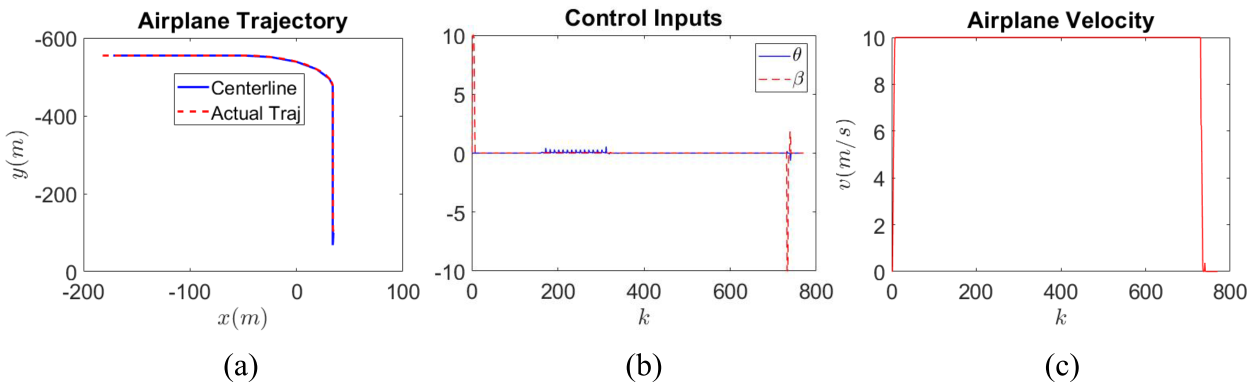

Fig. 9 The aircraft trajectory (a), control inputs (b), and velocity profile (c), for arrival simulation. "Turning" command is received before the aircraft approaches v_1. The airplane maintains the desired velocity and passes the node.

VIII. Conclusion

This paper develops a two-stage online action detection algorithm to temporally locate and classify actions from a streaming video sequence. The robustness of the approach have been shown to surpass the existing algorithms on

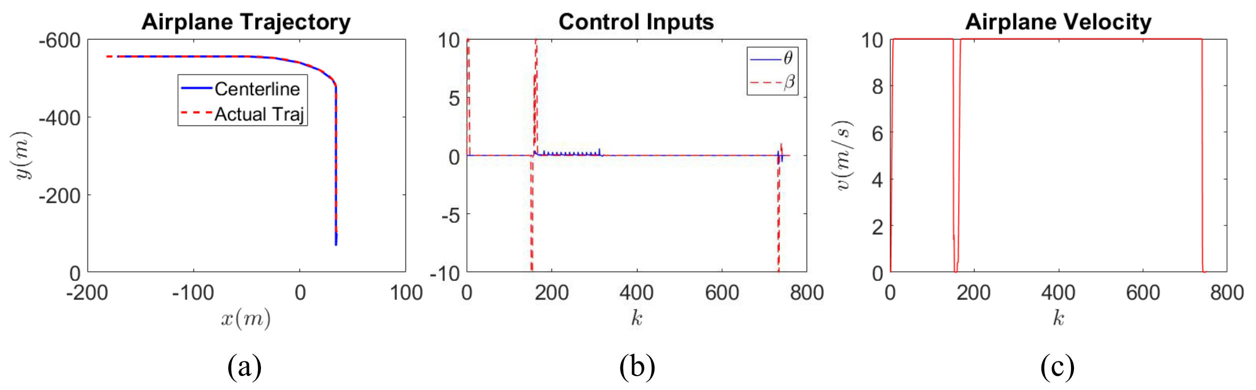

Fig. 10 The aircraft trajectory (a), control inputs (b), and velocity profile (c), for arrival simulation. "Turning" command is not received before the aircraft is within alarm distance d_a with v_1. The airplane drops to zero velocity at v_1 and restart once it receives a new command.

a real-world gesture dataset significantly. The algorithm is integrated with a control system to control a near-to-gate aircraft to cruise to a designated gate automatically. A hybrid controller is developed to switch between different cruising modes based on the gesture commands while tracking and following the reference lines. The effectiveness of the system is demonstrated by successfully conducting arrival experiments in a simulated airport.

Acknowledgments

This work is supported by the Office of Naval Research under award N00014-19-1-2266.

References

[1] Barbič, J., Safonova, A., Pan, J.-Y., Faloutsos, C., Hodgins, J. K., and Pollard, N. S., "Segmenting motion capture data into distinct behaviors," *Proceedings of Graphics Interface 2004*, Citeseer, 2004, pp. 185–194.

[2] Ji, S., Xu, W., Yang, M., and Yu, K., "3D convolutional neural networks for human action recognition," *IEEE transactions on pattern analysis and machine intelligence*, Vol. 35, No. 1, 2012, pp. 221–231.

[3] Simonyan, K., and Zisserman, A., "Two-stream convolutional networks for action recognition in videos," *Advances in neural information processing systems*, 2014, pp. 568–576.

[4] Carreira, J., and Zisserman, A., "Quo vadis, action recognition? a new model and the kinetics dataset," *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6299–6308.

[5] Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., and Paluri, M., "A closer look at spatiotemporal convolutions for action recognition," *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 6450–6459.

[6] Chao, Y.-W., Vijayanarasimhan, S., Seybold, B., Ross, D. A., Deng, J., and Sukthankar, R., "Rethinking the faster r-cnn architecture for temporal action localization," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1130–1139.

[7] Lin, T., Liu, X., Li, X., Ding, E., and Wen, S., "Bmn: Boundary-matching network for temporal action proposal generation," *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 3889–3898.

[8] Yang, X., Yang, X., Liu, M.-Y., Xiao, F., Davis, L. S., and Kautz, J., "Step: Spatio-temporal progressive learning for video action detection," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 264–272.

[9] De Geest, R., Gavves, E., Ghodrati, A., Li, Z., Snoek, C., and Tuytelaars, T., "Online action detection," *European Conference on Computer Vision*, Springer, 2016, pp. 269–284.

[10] Eun, H., Moon, J., Park, J., Jung, C., and Kim, C., "Temporal filtering networks for online action detection," *Pattern Recognition*, Vol. 111, 2021, p. 107695.

[11] Chen, C., Jafari, R., and Kehtarnavaz, N., "UTD-MHAD: A multimodal dataset for human action recognition utilizing a depth camera and a wearable inertial sensor," *2015 IEEE International conference on image processing (ICIP)*, IEEE, 2015, pp. 168–172.

[12] Liu, C., and Ferrari, S., "Vision-guided planning and control for autonomous taxiing via convolutional neural networks," *AIAA Scitech 2019 Forum*, 2019, p. 0928.

[13] UnrealEngine, "UnrealEngine," `https://www.unrealengine.com/en-US/what-is-unreal-engine-4`, 2018.

[14] Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E., and Sheikh, Y., "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields," *arXiv preprint arXiv:1812.08008*, 2018.

[15] Ding, J., Xiang, Y., Shen, L., and Tarokh, V., "Multiple change point analysis: Fast implementation and strong consistency," *IEEE Transactions on Signal Processing*, Vol. 65, No. 17, 2017, pp. 4495–4510.

[16] Scott, A. J., and Knott, M., "A cluster analysis method for grouping means in the analysis of variance," *Biometrics*, 1974, pp. 507–512.

[17] Bobick, A. F., and Davis, J. W., "The recognition of human movement using temporal templates," *IEEE Transactions on pattern analysis and machine intelligence*, Vol. 23, No. 3, 2001, pp. 257–267.

[18] Zhou, F., De la Torre, F., and Hodgins, J. K., "Aligned cluster analysis for temporal segmentation of human motion," *2008 8th IEEE international conference on automatic face & gesture recognition*, IEEE, 2008, pp. 1–7.

[19] Zhou, F., De la Torre, F., and Hodgins, J. K., "Hierarchical aligned cluster analysis for temporal clustering of human motion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 35, No. 3, 2012, pp. 582–596.

[20] Ng, A. Y., Jordan, M. I., and Weiss, Y., "On spectral clustering: Analysis and an algorithm," *Advances in neural information processing systems*, 2002, pp. 849–856.