Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jaka Cikač

# Upravljanje kvadrokopterja z gestami

MAGISTRSKO DELO

MAGISTRSKI PROGRAM DRUGE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: izr. prof. dr. Danijel Skočaj
Somentor: ass. prof. DI Dr. Friedrich Fraundorfer

Ljubljana, 2017

UNIVERSITY OF LJUBLJANA
FACULTY OF COMPUTER AND INFORMATION SCIENCE

Jaka Cikač

# Drone control using gestures

MASTER'S THESIS

THE 2ND CYCLE MASTER'S STUDY PROGRAMME
COMPUTER AND INFORMATION SCIENCE

SUPERVISOR: assoc. prof. dr. Danijel Skočaj
CO-SUPERVISOR: ass. prof. DI Dr. Friedrich Fraundorfer

Ljubljana, 2017

# Povzetek

**Naslov:** Upravljanje kvadrokopterja z gestami

Kvadrokopterji postajajo vse bolj priljubljeni in integrirani v današnjo družbo. Zmožni so visoko resolucijskega snemanja in avtonomnega navigiranja pri vrtoglavih hitrostih, navdušijo pa tudi kot vsakodnevna igrača. Kvadrokopterje je potrebno tudi nadzirati, za kar večinoma uporabljamo mobilne telefone. V tem delu smo razvili sistem za nadzor kvadrokopterja z gestami. Naš cilj je bil razviti sistem, ki bi se lahko izvajal na nizkocenovnem kvadrokopterju, ki je opremljen le z barvno kamero in zmogljivim vgrajenim računalnikom. Tak kvadrokopter smo tudi sestavili. Sistem je razdeljen v tri module - detekcija akcije z optičnim tokom, ocena človeške poze s konvolucijskimi nevronskimi mrežami ter klasifikacija geste z relacijskimi značilkami osnovanimi na človeški pozi. Integrirani sistem za nadzor kvadrokopterja z gestami smo implementirali s pomočjo knjižnice OpenCV in meta operacijskega sistema ROS. V namen razvoja in evalvacije sistema smo sestavili svojo bazo slik DS2017, v kateri je skupno 640 gest, ki jih je izvedlo 20 ljudi. V evalvaciji pokažemo, da sistem dosež zadovoljivo točnost pri detekciji akcij ter da hitro in natančno detektira človeško pozo in odlično klasificira detektirane geste.

## Ključne besede

*kvadrokopter, razpoznavanje gest, ocena človeške poze, optični tok, interakcija človek-robot, konvolucijske nevronske mreže, avtonomni letalnik*

# Abstract

**Title:** Drone control using gestures

Quadcopters are becoming more popular and integrated into modern society. From high resolution video recording to autonomous navigation at high speed, quadcopters even shine as an everyday toy. We are now familiar with controlling quadcopters via our mobile phones. In this work we set out to develop a quadcopter gesture control system. We aspired to develop a system that can be used on a low-cost quadcopter equipped with a simple RGB camera and a powerful embedded computer. We also assembled such a quadcopter. The system is split into three modules - action detection with optical flow, human pose estimation with convolutional neural networks and gesture classification with relational features computed on the human pose. The integrated system is developed with the help of OpenCV and meta operating system ROS. For the purpose of development and evaluation we also assembled our own dataset called DS2017, in which 640 gestures are performed by 20 people. We show that action detection can detect actions sufficiently well, the human pose estimation works very well at high speed and gesture classification achieves high accuracy.

## Keywords

*quadcopter, gesture recognition, human pose estimation, optical flow, human-robot interaction, convolutional neural networks, UAV*

“Where other men blindly follow the truth, remember...”

“*Nothing is true.*”

“Where other men are limited by morality or law, remember...”

“*Everything is permitted.*”

“We work in the dark to serve the light. We are *Engineers.*”

# Contents

# List of used acronyms

| **acronym** | **meaning** |
|---|---|
| **CA** | classification accuracy |
| **SVM** | support vector machine |
| **HOG** | histograms of oriented gradients |
| **SOTA** | state of the art |
| **PDF** | probability density function |
| **KDE** | kernel density estimation |
| **UAV** | unmanned aerial vehicle |
| **PC** | personal computer |
| **GPU** | graphical processing unit |
| **CNN** | convolutional neural networks |
| **NN** | neural networks |
| **CPM** | convolutional pose machines |
| **LSTM** | long short-term memory |
| **AUC** | area under the curve |
| **CPU** | central processing unit |
| **RAM** | random access memory |
| **GPS** | global positioning system |
| **MAV** | micro aerial vehicles |
| **NATOPS** | naval air training and operating procedures standardization |
| **J-HMDB** | joint-annotated human motion data base |
| **OF** | optical flow |
| **FMU** | flight management unit |

| | |
|---|---|
| **RC** | remote control |
| **SRAM** | static random access memory |
| **SoC** | system on a chip |
| **IO** | input / output |
| **ESC** | electronic speed control |
| **ARM** | Advanced RISC Machine |
| **PPM** | pulse position modulation |
| **PWM** | pulse width modulation |
| **LPDDR** | low power double data rate memory |
| **VOT** | visual object tracking |
| **BRW** | background ratio weighting |
| **RANSAC** | random sample consensus |
| **HDMI** | high definition multimedia interface |
| **L.K.** | Lucas and Kanade |
| **FPS** | frames per second |
| **mAP** | mean average precision |
| **AP** | average precision |
| **ASMS** | adaptive scale mean shift |
| **OKS** | object keypoint similarity |

# Razširjeni povzetek

## I Uvod

Kvadrokopterji postajajo vse bolj priljubljeni in integrirani v današnjo družbo. Najdemo jih v raziskovalnih laboratorijih, pri profesionalih snemalcih in fotografih, reševalnih skupinah in drugje. To ni presenetljivo glede na razne funkcije, ki jih lahko opravljajo. Od visoko resolucijskega snemanja do avtonomnega navigiranja pri vrtoglavih hitrostih, navdušijo pa tudi kot vsakodnevna igrača. Kvadrokopterje je potrebno tudi nadzirati, česar smo do sedaj vajeni z uporabo mobilnih telefonov ali pa z upravljalnim daljnicem. V tem delu se lotimo nadzora kvadrokopterja na drugačen način in sicer z uporabo gest, ki jih kvadrokopter prepozna s pomočjo barvne kamere in metod računalniškega vida.

## II Kratek pregled sorodnih del

Poskusov upravljanja kvadrokopterjev s pomočjo gest je bilo že precej. Glavne omejitve so se pokazale v računski moči, ki je potrebna na kvadrokopterju in pa zmogljivost video sistema. Zaradi teh omejitev je večina do sedaj razvitih sistemov uporabljala zunanje naprave za upravljanje, kar vključuje zunanji PC in pa na primer Microsoft Kinect v kombinaciji s kvadrokopterjem Parrot AR [1].

Podoben pristop za upravljanje kvadrokopterja uporablja napravo za zaznavanje gibanja Leap Motion [2], ki prevede premike v geste ter jih posreduje kvadrokopterju kot ukaz. Zelo soroden pristop je tudi uporaba rokavic s pospeškometri, kot na primer [3]. Ta način upravljanja kvadrokopterja je zelo intuitiven in ne potrebuje veliko računske moči, zahteva pa uporabo specifičnih rokavic ter konstantno pozornost uporabnika. To se je kmalu preneslo tudi na vedno bolj priljubljene pametne ure, kot v primeru [4], kjer se kvadrokopter upravlja z uro Apple Watch.

Raziskovalci so v [5] implementirali nadzor cenovno dostopnega kvadrokopterja Parrot AR z gestami s pomočjo barvne kamere, tako da so sledili obrazu in dlani uporabnika. Ta pristop je sicer zahteval, da uporabnik nosi posebne pobvarvane rokavice, celoten sistem pa se je izvajal na ločenem računalniku.

## II.I Detekcija akcij

S pomočjo detekcije akcije želimo lokalizirati akcijo, ki se izvaja v določenem časovnem okviru na videu. Presentljivo malo raziskav se osredotoča na detekcijo akcije v realnem času, kjer je potrebno zaznati akcijo v neprekinjem videu. Do te ugotovitve so prišli tudi avtorji [6], ki so ugotovili, da za ta primer ni na voljo primerne baze podatkov, s pomočjo katere bi lahko ocenili delovanje različnih metod. Zato so predstavili svojo bazo podatkov na kateri so nato primerljali najnovejše metode za detekcijo akcij in ugotovili, da nobena izmed obstoječih metod ni dovolj dobra za rešitev tega problema in pa da nobena ni dovolj hitra.

Najnovejše metode za detekcijo akcije imajo poudarek na lokalizaciji akcije v smislu pozicije na sličicah v videu namesto v času, kar dosežejo s pomočjo konvolucijskih nevronskih mrež. Ena izmed takih metod je "Action Tubes" [7], ki združuje detekcijo akcij na sličici s klasifikacijo detektirane akcije.

Kljub napredkom metod, ki so osnovane na konvolucijskih nevronskih mrežah v smislu natančnosti, so le te še vedno precej počasne in zelo kompleksne.

## II.II Ocena človeške poze in klasifikacija gest

Prepoznavanje oziroma klasifikacija akcij se od detekcije razlikuje v vhodu in izhodu. Medtem ko je izhod detekcije akcije pozicija na sliki ali pozicija v času, je izhod klasifikacije akcije poimenovana dotična akcija, ki se izvaja.

Prvo vprašanje, ki smo si ga zastavili je, če je klasifikacija gest osnovana na oceni človeške poze uspešna, kar je bilo tudi vprašanje avtorjev [8]. Avtorji so primerjali nizko-nivojske značilke (osnovane na optičnem toku) in visoko-nivojske značilke (ocena človeške poze). Uporabili so relacijske značilke človeške poze, ki so bile predstavljene v [9]. Ugotovili so, da je klasifikacija gest na osnovi človeške poze precej uspešnejša in tudi, da uporaba obojih značilk hkrati ne pripomore veliko.

Podobne rezultate so dosegli tudi avtorji [10]. Za potrebo evalvacij algoritmov pa so tudi skovali bazo podatkov JHMDB, kjer so na posnetkih ljudi, ki izvajajo različne akcije, anotirali človeško pozo ter video posnetke označili z akcijami. Avtorji so ugotovili, da je za visoko točnost klasifikacije gest potrebno uporabiti značilke osnovane na človeški pozi.

Seveda se je tudi ocena človeške poze poslužila konvolucijskih nevronskih mrež, kjer so pionirji [11] izkoristili časovno komponento videa in propagirali oceno poze skozi več sličic čez celoten posnetek. Njihova metoda je takrat dosegla boljše rezultate od tradicionalnih pristopov, kot so ″Poselets″ [12] in naključni gozdovi.

Hitrost in natančnost so avtorji [13] še izboljšali s sekvenčnim predikcijskim modelom ″Convolutional Pose Machines″ (oz. CPM), kjer se 2D verjetnostna mapa propagira čez več faz. Ta metoda je dosegla izjemno visoko natančnost in postavila nove standarde na znanih zbirkah podatkov, kot so MPII in FLIC. Žal pa je bila hitrost te metode še vedno prepočasna za delovanje v realnem času.

Z ozirom na metodo CPM so nato avtorji [14] uspešno razvili metodo ″Multi-Person 2D Pose Estimation using Part Affinity Fields″ oziroma MPE-PAF, ki deluje v realnem času. Metoda skupaj s predikcijo lokacije točk na človeškem skeletu za izhod napove tudi povezavo med posameznimi točkami, s katero se direktno tvori predstavitev človeške poze.

## III Predlagana metoda

Za upravljanje kvadrokopterja z gestami ne želimo uporabiti posebnih zunanjih naprav, kot so običajno počeli ostali, pač pa želimo imeti interakcijo neposredno s kvadrokopterjem. To omogoča uporabo kvadrokopterja, ki je opremljen le z barvno kamero in dovolj zmogljivim računalnikom. Predlagana metoda tako zahteva le barvni video, na katerem najprej zaznamo, da uporabnik izvaja akcijo s pomočjo optičnega toka, zaznano akcijo pa nato posredujemo v algoritem za oceno človeške poze. V ta namen uporabimo konvolucijske nevronske mreže avtorjev [14]. Po zaznani pozi uporabnika, se pozicije različnih točk prenesejo v algoritem za klasifikacijo geste. Za klasifikacijo geste najprej izračunamo različne geometrične značilke na ocenjeni pozi, kar pa s pomočjo algoritma vreče besed pretvorimo v histogram značilnih točk geste, ki ga nato posredujemo v metodo podpornih vektorjev, s katerim klasificiramo dobljeni histogram v gesto.

### III.I Detekcija akcije

Detekcija akcije je sestavljena iz relativno preprostih metod. Najprej se poslužimo detekcije osebe, za njeno lokalizacijo na trenutni sličici, nakar osebi sledimo z uporabo izredno hitrega in robustnega kratkoročnega sledilnika ASMS.

Osebo med sledenjem ponovno zaznamo vsakih 30 sličic, da preprečimo možno izgubo natančnosti sledenja zaradi akumulacije napake ali nepričakovanega premika kvadrokopterja. Po lokalizaciji osebe med dvema zaporednima sličicama izračunamo optični tok v okolici osebe z metodo Lucas-Kanade.

Optični tok nato sfiltriramo s pomočjo algoritma RANSAC, s tem pa se znebimo optičnega toka, ki se pojavi zaradi premikanja kvadrokopterja. Po filtriranju nam ostane le optični tok, ki ga povzroča oseba.

Po filtriranju optičnega toka preštejemo število točk, ki ostanejo v petih zaporednih sličicah, ki nato tvorijo skupek. Ta skupek nato klasificiramo z metodo podpornih vektorjev z linearnim jedrom, katere rezultat nam pove, če skupek vsebuje akcijo ali ne. Skupki so nato poslani v *krožni pomnilnik*, s pomočjo katerega lahko ponavljamo detekcijo akcije v nedogled na neprekinjenem videu, hkrati pa jih lahko uporabimo za filtriranje akcij, ki so morda napake pri filtriranju optičnega toka. Zaradi enostavnosti in hitrosti posameznih modulov je zgoraj opisana metoda izredno hitra.

## III.II Ocena človeške poze

Za oceno človeške poze smo izbrali metodo "Multi-Person 2D Pose Estimation using Part Affinity Fields" oziroma MPE-PAF [14], ki je zmožna oceniti pozo večih ljudi v realnem času. Metoda se močno opira na metodo CPM (ang. "Convolutional Pose Machines), kjer so avtorji uporabili napredne konvolucijske nevronske mreže, katerih rezultat je 2D mapa verjetnosti za pojavitev posameznega dela telesa. Te mape verjetnosti so poslane skozi več faz, kjer vsaka faza izboljša natančnost prejšnje faze. Nato pa se med seboj poveže različne točke na telesu in tvori predstavitev človeškega skeleta. Slabost metode CPM je, da mora biti skombinirana z detektorjem oseb, kar močno vpliva na natančnost. Slabost, ki je relevantna za naš primer pa je njena počasnost, saj lahko traja tudi več deset sekund, da se oceni poza na eni sliki.

Metoda [14] uporablja več stopenjsko konvolucijsko nevronsko mrežo (VGG-19), katere vhod je barvna slika. Prvi del nevronske mreže izračuna verjetnostno mapo lokacij točk človeškega skeleta in pa povezovalno polje, ki nosi informacijo o povezavi delov telesa. To dvoje je skupaj posredovano v naslednje stopnje, kjer vsaka še izboljša natančnost. Skupna inferenca točk človeškega skeleta in povezave med deli izredno povišata hitrost metode. Poleg tega pa ta metoda ne potrebuje ločenega detektorja oseb in je tako neodvisna od delovanja le-tega, zmanjša pa se tudi število napačno zaznanih oseb.

## III.III Klasifikacija geste

Za klasifikacijo gest uporabimo značilke osnovane na ocenjeni človeški pozi. Specifično se poslužimo relacijskih značilk človeške poze, katere označimo s "Pose Features". Izračunamo pet različnih tipov takih značilk in sicer

1. pozicija,
2. razdalja,
3. orientacija,
4. kot med točkami in
5. razlika med točkami v času.

Nato uporabimo metodo vreče besed, za kar je najprej potrebno zgraditi slovar besed. V ta namen se vsaka gesta razdeli na posamezne sličice, kjer na vsaki sličici izračunamo zgoraj navedene značilke glede na ocenjeno človeško pozo, kar nam vrne deskriptor za sličico. Število deskriptorjev je odvisno od dolžine geste v smislu števila sličic. Nato izvedemo gručenje deskriptorjev z algoritmom K-means za kvantizacijo le-teh v posamezne gruče, s čimer dobimo centre posameznih gruč. Vsaka gruča je nato predstavljena kot celica v histogramu. Za vsak deskriptor se izračuna razdalja do vsakega izmed centrov nato pa se poveča celica gruče, ki pripada najbližjemu centru. Tako pridelamo histogram, ki predstavlja celotno gesto. Ta histogram je nato normaliziran za neodvisnost od dolžine geste. Histogrami so zbrani v zbirko in tvorijo značilke za metodo podpornih vektorjev z radialnim jedrom, s pomočjo katere nato klasificiramo geste in tako pridemo do končne predikcije, ki se prevede v ukaz za kvadrokopter.

## IV Implementacija

Sistem je implementiran v programskem jeziku C++ s pomočjo odprtokodne knjižnice OpenCV in meta operacijskega sistema ROS, ki nudi modularno procesiranje skupaj s sporočilnim sistemom. Tako si lahko različne računske enote med seboj pošiljajo sporočila, čeprav so porazdeljene med več računskih sistemov. S pomočjo ROS lahko uporabimo tudi ločen računalnik, v katerem je nameščena grafična kartica, ki je potrebna za izračun človeške poze v realnem času, medtem ko se ostale komponente sistema izvajajo na kvadrokopterju. ROS nam prav tako omogoča neodvisnost od naše trenutne mobilne platforme.

Prvo komponento naše implementacije tvori *VideoNode*, ki upravlja s kamero in nam posreduje potrebne barvne sličice s pomočjo kamere na kvadrokopterju. Ti podatki so prenešeni v enoto za detekcijo akcije, ki preveri če se na zaporednih sličicah nahaja akcija. V primeru, da se, se sličice posredujejo v enoto za oceno človeške poze, ki se izvaja na ločenem računalniku z grafično procesno enoto proizvajalca NVIDIA. Naloga enote za oceno človeške poze je izračun pozicije ključnih točk skeleta človeka, s pomočjo konvolucijskih nevronskih mrež. Te pozicije se nato prenesejo v enoto za klasifikacijo geste, ki izračuna značilno geometrično konfiguracijo človeške poze nato pa s pomočjo metode podpornih vektorjev klasificira dobljene značilke v gesto. Gesta je nato posredovana kvadrokopterjevemu avtopilotu kot ukaz, ki ga nato kvadrokopter izvede. Ko je ukaz izveden se vrnemo v začetno stanje in celoten postopek se ponovi.

# V Eksperimentalna evalvacija

## V.I Uporabljene baze podatkov

Za razvoj metod in njihovo evalvacijo smo uporabili dve bazi podatkov in sicer JHMDB ter DS2017. Prva je zelo znana baza podatkov za oceno klasifikacije akcij ter oceno človeške poze. Sestavljena je iz 920 videov, zbranih s spletnega portala YouTube. Celotna baza vključuje 23 različnih akcij.

DS2017 smo sestavili in posneli za potrebe magistrske naloge. Vključuje

4 različne geste za nadzor kvadrokopterja (levo, desno, gor, dol), ki so predstavljene na 2 različna načina. Prvi vključuje 4 enostavnejše (nadzorovane) geste, ki smo jih specificirali sami, drugi pa 4 intuitivne geste, za katere uporabniki niso prejeli natančnih navodil. Oba načina sta bila posneta tudi s stabilno kamero ter z nestabilno kamero, za namen simulacije nenadnih premikov kvadrokopterja iz različnih razlogov. DS2017 vključuje 20 različnih oseb, ki so skupaj izvedle 640 gest.

## V.II Evalvacija detekcije akcij

Algoritem za detekcijo akcije smo ocenili na različnih kategorijah baze podatkov DS2017. Ugotovili smo, da so akcije največkrat uspešno zaznane, ko uporabnik izvaja nadzorovane geste. Akcija se smatra kot zaznana, če algoritem uspešno oceni, da se v nekaj zaporednih skupkih (kjer vsak vsebuje 5 sličic) nahaja akcija. V tem primeru algoritem za detekcijo akcij zazna akcijo v **85,93%**. Intuitivne geste je težje zaznati in sicer je detekcija akcij v tem primeru uspešna na **78,12%** primerov.

Ugotovili smo tudi, da je detekcija akcije bolj uspešna, če kamera ni stabilizirana, saj se takrat akcije zdijo bolj izrazite (proizvedejo več optičnega toka).

Največ težav se pojavi pri detekciji intuitivne geste "gor", saj se izvaja spredaj pred trupom uporabnika, kar naredi akcijo oziroma pozo človeka težje razločljivo od ozadja.

## V.III Evalvacija ocene človeške poze

Evalvacije ocene človeške poze smo se najprej lotili s povzetkom rezultatov, ki so jih dosegli avtorji metode [14]. Metoda je bila najbolje uvrščena na izzivu COCO 2016 Keypoints in prav tako dosegla najvišje rezultate na bazi podatkov MPII za oceno človeške poze v letu 2016. Avtorji so izmerili tudi hitrost delovanja metode in ugotovili, da se metoda v povprečju izvaja kar za 6 magnitud hitreje, kot ostale metode uvrščene kot najboljše na bazi podatkov MPII.

Metodo smo kvalitativno ocenili tudi na naši bazi slik DS2017, kjer smo ugotovili, da je največ težav z oceno poze pri intuitivni in nadzorovani gesti "dol", saj uporabnik takrat prekriža obe roki pred svojim telesom. Prav tako smo našli težave z oceno poze, kadar se okoli uporabnika pojavijo močne sence ali pa ko so sličice premalo osvetljene. Ugotovili smo tudi, da nestabilna kamera oziroma premikanje kvadrokopterja ne vpliva na kvaliteto ocenjene poze, kljub navideznim rotacijam uporabnika v prostoru.

Ugotovili smo torej, da je metoda [14] izredno dobra za ocenjevanje človeške poze, saj je le ta ocenjena v večini primerov gest v DS2017. Presenetljivo je bila poza ocenjena pravilno tudi pri gesti "gor", ko uporabnik popolnoma prekrije zgornji del rok s spodnjim delom.

Prav tako je metoda zelo hitra, saj je zmožna oceniti človeško pozo v realnem-času kar za 19 ljudi hkrati s hitrostjo 8,8 sličic na sekundo, če je uporabljena mobilna grafična procesna enota NVIDIA GeForce 1080-GTX. V naših testih smo ugotovili, da je metoda zmožna oceniti človeško pozo za eno osebo z 12,3 sličicami na sekundo, če uporabimo novejšo grafično procesno enoto NVIDIA GeForce 1080-GTX Ti.

## V.IV Evalvacija klasifikacije gest

Klasifikacijo gest smo ocenili na bazah podatkov JHMDB in DS2017. JHMDB vključuje realistične posnetke, ki so jih avtorji zbrali s portala YouTube in posledično so akcije izredno zahtevne zaradi njihove različnosti.

JHMDB je sestavljen iz 21 različnih akcij. Algoritem za klasifikacijo gest osnovan na človeški pozi doseže klasifikacijsko točnost **57,08%**.

Za primerjavo je trenutno najboljša metoda dosegla klasifikacijsko točnost 71,08%, osnovana pa je na konvolucijskih nevronskih mrežah, zaradi česar je precej počasnejša od metode, ki jo uporabljamo. Potrebuje namreč več kot 220 ms na sličico, medtem ko naša metoda potrebuje 54 ms za celotno gesto (25 sličic).

Na bazi podatkov DS2017 dosežemo precej dobre rezultate. Vhod v

DS2017 klasifikacijska točnost.

| **Kategorija** | **CA** [%] |
|---|---|
| Nadzorovane geste | 97,75 |
| Stabilne nadzorovane geste | 96,77 |
| Nestabilne nadzorovane geste | 97 |
| Intuitivne geste | 97,25 |
| Stabilne intuitivne geste | 96,5 |
| Nestabilne intuitivne geste | 96,4 |

**Tabela 2:** Klasifikacijska točnost (CA) za bazo podatkov DS2017, prikazana gleden na kategorijo.

algoritem klasifikacije gest so ocenjene točke posamezih delov telesa, ki se izračunajo na sličicah v videu, na katerih se izvaja gesta. Za vsako kategorijo so rezultati povzeti v Tabeli 2. Klasifikacijska točnost je izračuna kot povprečje za štiri različne razdelitve celotne množice primerov na učno in testno množico. Za kategoriji "nadzorovane" in "intuitivne" geste smo združili posnetke stabilne in nestabilne kamere. Najvišjo točnost dosežemo v kategoriji z nadzorovanimi gestami, saj so te lažje in bolj izrazite. Zaradi težav z oceno poze, ko se prekrižajo roke se posledično zniža tudi klasifikacijska točnost za gesto "dol". Veliko razlike med stabilno in nestabilno kamero nismo našli. Visoka klasifikacijska točnost v kategoriji z intuitivnimi gestami nam pove, da se model dobro nauči tudi intuitivnih, bolj komplesnih, gest.

## V.V Evalvacija integriranega sistema

Za evalvacijo integriranega sistema smo uporabili testno množico baze podatkov DS2017. V tem primeru nas zanima klasifikacijska točnost geste čez celoten sistem. Tako se video najprej pošlje čez detekcijo akcije, ki posreduje sličice z akcijo v oceno človeške poze nato pa se izračuna značilke in gesto klasificira. V ta namen smo model naučili na združeni učni množici, ki vključuje intuitivne in nadzorovane geste. Kot do sedaj smo evalvacijo razdelili v kategorije glede na stabilno in nestabilno kamero.

Algoritem za detekcijo akcije je uspešno zaznal akcijo v **83,13%** primerih, kar je 133 od 160 posnetkov z gestami. Evalvacijo nadaljujemo na posnetkih, kjer je bila akcija zaznana. Opazili smo, da algoritem za detekcijo akcije občasno odreže prvih 5 sličic akcije, saj takrat ni zaznana. Kljub temu, to ni močno vplivala na končno klasifikacijo.

Klasifikacijska točnost je podobna za nadzorovane in intuitivne geste. Za geste "gor, dol" in "desno" je klasifikacija uspešna v **100%** primerov. Gesta "levo" pa je v redkih primerih zamenjana za desno. Možen vzrok tega je njujna podobnost.

Če je detekcija akcije uspešna, je torej velika verjetnost, da bo tudi končna klasifikacija geste pravilna. Možen vzrok za nižjo uspešnost v samostojni evalvaciji klasifikacije gest so akcije, ki niso uspešno zaznane v fazi detekcije akcije, kjer le te niso dovolj izrazite (v smislu gibanja ali pa prekrivanja) ali pa so nenavadne ali prehitre.

Celoten sisteme se v povprečju od zaznane akcije do končne klasifikacije izvede v **2,14 sekundah**. Večino tega časa se porabi za oceno človeške poze, saj le ta potrebuje v povprečju *2,03 sekund* za 25 sličic, ki vsebujejo akcijo. Detekcija akcije se izvaja v realnem-času in povprečno porabi *58 milisekund* na sličico. Prav tako izračun značilk in končna klasifikacija geste deluje v realnem-času in sicer porabi v povprečju *54 milisekund*.

Meritve so bile izvedene na računalniku s procesorjem Intel Core i7 4770K, kar pomeni da bi se celoten sistem na kvadrokopterjevem računalniku izvedel počasneje. Kljub temu časi ne bi bili veliko daljši za detekcijo akcije in klasifikacijo geste, ki pa sta izredno hitri komponenti. Čas za oceno poze pa je odvisen od grafične kartice, ki se uporablja na zunanjem računalniku in se tako ne bi podaljšal.

# VI Sklep

V tem delu smo razvili realno-časovni sistem za nadzor kvadrokopterja. Najprej smo se izobrazili o preteklih raziskavah na področju in ugotovili, da večina dosedanjih sistemov zahteva uporabo zunanjih naprav ali pa izredno dragih kvadrokopterjev.

Naš cilj je bil razviti sistem, ki bi bil uporaben na nizkocenovnem kvadrokopterju, ki je opremljen le z barvno kamero in zmogljivim vgrajenim računalnikom. V ta namen smo tak kvadrokopter tudi sestavili.

Razvili smo sistem s tremi moduli, ki najprej zazna akcijo s pomočjo detektorja oseb, zelo hitrega kratkoročnega sledilnika in optičnega toka. Po detekciji akcije, uporabimo napredno metodo [14] za oceno človeške poze v realnem času na sličicah, ki vsebujejo akcijo. Po izračunu točk človeške poze, izračunamo relacijske značilke ter jih sestavimo v deskriptor, ki ga s pomočjo metode podpornih vektorjev klasificiramo, izhod pa vzamemo kot končno napovedano gesto.

Integrirani sistem za nadzor kvadrokopterja z gestami smo implementirali s pomočjo odprtokodne knjižnice OpenCV in meta operacijskega sistema ROS, ki nam omogoča porazdeljeno izvajanje ter komunikacijo med moduli. To je zelo pomembno, saj smo zaenkrat prisiljeni izvajati oceno človeške poze na zunanjem računalniku s specifično grafično procesno enoto. Taka implementacija pa nam omogoča enostavno izvajati vse dele na kvadrokopterju, ko bo strojna oprema to omogočala. Prav tako smo zasnovali detekcijo akcij na osnovi krožnega pomnilnika, kar nam omogoča kontinuirano izvajanje.

V namen razvoja in evalvacije sistema smo sestavili svojo bazo podatkov DS2017, v kateri je skupno 640 gest, ki jih je izvedlo 20 ljudi.

Evalvacija detekcije akcij je pokazala, da zaznamo akcijo v 83% primerov, bolj uspešni pa smo z detekcijo nadzorovanih akcij. Metoda za oceno človeške poze [14] je zmožna oceniti pozo izredno natančno, do napak pride le pri prekrižanju rok pri gesti “dol”. Klasifikacija gest doseže visoko točnost in sicer 96,8% na bazi podatkov DS2017.

Celoten sistem v povprečju potrebuje **2,14** sekunde, da pošlje ukaz kvadrokopterju od trenutka, ko je bila akcija zaznana.

## VI.I Prihodnje delo

Del sistema, ki nam preprečuje resnično realno-časovno delovanje in izvajanje celotnega sistema na kvadrokopterju je ocena človeške poze. Čeprav je metoda, ki jo uporabljamo izredno hitra, še vedno potrebuje 2 sekundi za oceno poze na gesto. Ta čas pa lahko dosežemo samo z izvajanjem na zunanji grafični procesni enoti. Rešitev je več.

Če ogrodja za uporabo konvolucijskih nevronskih mrež začnejo podpirati ostale proizvajalce in grafične procesne enote tipa ARM, lahko izničimo potrebo po specifični grafični procesni enoti proizvajalca NVIDIA.

Druga rešitev je, da uporabimo vgrajeni računalnik NVIDIA Jetson TX 1 ali NVIDIA Jetson TX 2 na kvadrokopterju. Le ta ima na voljo 256 jeder CUDA, kar je dovolj za izračun poze, ne bi pa bilo dovolj hitro. Kljub temu bi to bil korak v pravo smer.

V tem primeru bi naš sistem lahko enostavno popolnoma izvajali na kvadrokopterju, zahvala pa gre implementaciji v sistemu ROS, ki omogoča izvajanje neodvisno od arhitekture strojne opreme.

Čas ocene poze bi lahko znižali tudi z implementacijo interpolacije ocene točk čez več sličic. Tako bi lahko dejansko pozo ocenili le na nekaj sličicah, ostale pa bi imele točke človeškega skeleta interpolirane.

V nadaljevanju bi bilo zanimivo dodati tudi več različnih gest tako v bazo podatkov DS2017, kot tudi v naš sistem za nadzor kvadrokopterja z gestami.

# Chapter 1

# Introduction

## 1.1 Motivation

Drones have become more popular in the last decade, as research and development pushes the boundaries of what they are capable of doing. They are being used in search and rescue operations, military operations, exploration tasks, formation flights, delivering mail and food and there are even other clever ideas, such as using the drones for emergency defibrillator delivery.

As quadcopters get smaller and more affordable there are many new possibilities for their use, reaching a wider audience. It is not uncommon to find drones at many university laboratories, research institutes and now even normal households. This opens up the possibility of making drones a part of people's lives to perform simple tasks or simply be very interesting toys. An example of such a quadcopter is shown on Figure 1.1. There are countless enthusiasts using drones equipped with the latest video capturing technology to provide stunning aerial views. Whatever the use of the drone, there is always a much needed component - *control*. Without the ability to control drones in a safe and efficient manner they are not really useful. There are now various ways of controlling the drones and the most popular ones are autopilot control (usually done with way-point flying using GPS), manual flight (done with remote controllers as used for decades in the hobbyist communities), and lately, since every person now owns a smart-phone, mobile applications that are used for this purpose. To enhance the experience manufacturers are even using virtual reality to provide first person views of drone's flight using special goggles and on-board cameras.

**Figure 1.1:** DJI Spark, an example of a small sized quadcopter. Image source: www.dji.com

This work will focus on controlling the drones in a different way. As the title suggests, we wish to control the drone by using gestures. Hands free control of the drone would enable the operators to focus more on what they are doing, be it an activity that requires their physical engagement or rather just enjoying the scenery.

Since most micro drones are not capable of on-board processing with a dedicated computer the focus of this thesis is not on micro aerial vehicles but small to medium ones, that can carry enough payload to have a dedicated on-board computer and a camera. This enables us to use computer vision algorithms that are required for user detection, action recognition and more. Such computer vision methods have been greatly explored in the past and have come to a point where they can be run on consumer hardware in real-time. Together with the appropriate hardware - namely the small to medium sized drones - and well developed computer vision methods we are able to design a system that allows interacting with drones by gestures without necessitating external stationary devices or specialized computer systems.

## 1.2 Problem definition

There are plenty of ways of controlling drones via separate devices using either RGB-D cameras mounted on laptops, using Microsoft Kinect, Leap Motion, custom-made gloves with motion sensors, or even using an Apple Watch by performing gestures that are picked up by motion sensors and gyroscopes. However, there are situations when the operator has to interact with the drone directly. As such, being able to control a drone with gestures would be an improvement in environments such as security patrolling, manufacturing grounds, sports tracking and others.

In this master's thesis we set out to lay the ground work on how drones could be controlled using full-body gestures without any of those devices. Instead, the system is computer vision based.

In order to be able to "see" the gestures, a drone should be equipped with an RGB monocular camera, from which a video feed of the user can be processed using computer vision, to estimate the human pose and further recognize and classify the gestures that are performed. The drone should then be able to interpret the gestures as commands for some basic actions such as landing, taking off and others. This would allow a drone to be controlled without the use of extra devices, with which a user is required to control a drone, such as Microsoft Kinect or custom-made devices.

In order to create a system for controlling a drone with gestures there should be some requirements. An ideal implementation should run on the drone itself. This is not trivial, as most drones are equipped with mobile or embedded processors and computer vision algorithms typically require a lot of processing power. The user should also feel safe while controlling the drone, so the drone has a restricted movement space, or in other words, there should not exist a gesture that puts the user in danger.

Since the system will use human pose estimation and optical flow as inputs, these two components would need to be implemented on-line and in real-time. Most current research for human pose estimation in videos is barely capable of running real-time on powerful GPUs and even that for a single image.

## 1.3 Contributions

The contribution of this work will be the implementation of a system that combines processing on a drone and on a separate PC, using a GPU for processing human pose estimation, together with optical flow based motion estimation and classification of the user's gestures running on the drone itself.

The need for an external PC is only due to not having appropriate hardware on the drone yet available, however we describe how to eliminate this need in later chapters.

Since a drone is a moving platform it creates optical flow by moving itself around, therefore the system should be able to subtract optical flow created by these movements in order to isolate the optical flow created by the movement of user's hand, from which the gestures are then classified.

Therefore a theoretical contribution of this thesis is a working system that is able to visually classify gestures based on human pose estimation and optical flow and use this as commands for a drone. Technical aspect of the thesis is the system's integration on the drone, which requires it to be computationally efficient. This will enable gestures to be used for control, whilst using only an RGB camera instead of the depth camera and requires no additional devices between the user and the drone.

## 1.4 Structure

This work is structured into **seven** chapters. In introductory **Chapter 1** we learned about our motivation for this work and its contributions.

In **Chapter 2** we examine approaches and research that was carried out in related fields of computer vision so far. We give an insight into state of the art methods and how this enables us to combine some promising works into a real-time implementation.

In **Chapter 3** we examine the potential mobile platform for the system implementation. We first evaluate and discuss the selection of necessary components that are required and then describe how we built our own quadcopter that suits our needs. At the end of this chapter the reader will know which components make up a drone and how it can be used as a platform for many computer vision enabled tasks.

**Chapter 4** is a detailed presentation and description of computer vision methods used for gesture controlling drones. First we start with a general description of the proposed system and announce its separate phases. We then describe each phase separately and explain why we chose each method. We describe in detail the action detection pipeline, methods that are used for real-time pose estimation and finally gesture classification.

**Chapter 5** has a more practical overview of the system introducing the Robot Operating System (ROS), with which we have implemented the integrated system and how it offers us the ability to communicate and control the drone. Some implementation details are also discussed.

In **Chapter 6** we present the existing datasets that aided in development and then introduce our own dataset, with which we have the ability to evaluate the system as a whole. Collecting our own dataset allows us to specialize the system to a specific set of gestures. We also discuss the choice of gestures that make up this dataset. We then evaluate action detection, pose estimation and gesture recognition modules individually. Finally the whole integrated system is evaluated.

In the final **Chapter 7** we sum up our work and discuss possible future work that can improve the system in a way that it would require even less processing time, eliminates the need for an external PC and adds more gestures into the gesture set.

# Chapter 2

# Related Work

## 2.1 Gesture control

The field of gesture recognition has, in recent years, been explored to many depths, which can be seen from the many surveys that have appeared, such as [15] [16] [17]. And research on this topic is not slowing down. We, as humans, achieve many things with gestures and it is only natural that at some point we would like computers to recognize actions. In this way we can learn about human interaction with the world, recognize intent where we do not have the ability to record voice or the voice is hard to be heard, in which case we can clarify it with gestures. It is not uncommon to even train dogs to recognize gestures as commands instead of voice commands. It also allows people with hearing impairments to communicate. Of course it is very convenient to teach robots how to recognize gestures, so that they too can understand our intent or receive our commands better. Gestures have been used in popular movies as well, where characters like Tony Stark [18] interact with the whole room simply by waving hands.

Gesture recognition gives us the ability to convey some intent or information to the system that we wish to interact with. It is however very important to understand what a gesture is. This is not a simple task since there are many ”levels” of gestures, and the term is very ambiguous. A definition of a gesture really varies from one field of science to the other. We can create gestures using facial expressions, hands, arms or even the whole body. In this work we focus on gestures where the user uses the whole body.

## 2.2 Gesture controlling drones

Gesture control on drones has been attempted many times and it is clearly not an easy task. There has been some research done on how a person could control drones using gestures and there were some ”guidelines” written for it.

One such example is [19]. In their work authors researched, which gestures seem natural for users to interact with the drone and what kind of modality (gestures or voice) the users used. They set up a study where a person was controlling the drone according to the gestures that the users performed. The most recurring gestures were selected as the most natural ones. They found that users used the same gestures as people use for conveying information to their pets or even interpersonal gestures such as come here, point to precise location, come closer, stop, move left and right. They found that some gestures were more easily conveyed using voice such as fly sideways and land, where gestures like take a selfie or stop and come closer were dominantly conveyed using body gestures and less by using sound. One important result of the study is also that users were comfortable controlling the drone using gestures and even let the drone come closer to them as deemed safe by the researchers. They also learned that users required a gesture that tells the drone to perform emergency landing.

Another more recent example study also provides some ”guidelines” on which gestures make sense for controlling drones. They focus on users interfacing with a drone by means of non-traditional modalities such as gestures, speech and gaze direction [20]. Similar to the previously mentioned work, authors believe it is crucial for the user to not need constant attention and guidance of the drone, like it is required via a remote controller, in order to take the workload off the operator. It is precisely for this reason that they recommend utilization of natural and intuitive interaction techniques. In their work they evaluate different intuitive gestures and decide on three classes of ”mental models”, namely imitative, instrumented and intelligent. However this is not important here, as we are designing a system that implements gesture control and is relatively independent of the gesture set selected. However, they too, divide gestures into three main categories, hand control, upper / whole body control and gaze control. We focus the system on upper body control, which allows the drone to be further away from the user.

So it seems that users indeed like to control drones using gestures and they even agree on a set of gestures that would be acceptable for controlling the drones. There is also a lot of interest in gesture control since it takes away the need of constant attention. Why have we not completely adapted gesture control yet and still use remote controllers and mobile applications to do it?

The main limitations of gesture control on drones comes from computational power required on the on-board processor and a capable camera system. For that reason most gesture control systems that were developed, or researched, required external devices and an offline computer that could run the recognition algorithms. One of such examples is [1] where authors used a Parrot AR Drone and Microsoft Kinect to evaluate different metaphors for conveying controls for a variety of flying operations supported by the UAV.

This approach requires both an external computer and a Microsoft Kinect device, since the Parrot AR Drone is in no way capable of carrying such a heavy camera and an on-board computer. Authors did however find that users preferred 3D spatial interaction with the drone over the smartphone application that is bundled with the drone to control it. They also found that standing and performing whole body gestures was more accurate from users perspective, since incorrect commands are less likely to occur, as well as in terms of recognition, than gestures performed while seating or using only a hand.

Another approach, that uses a different external device, for example is [2]. Here the authors use a motion controller (namely LEAP motion) to detect the movements of hands which are then translated to gestures, which their drone understood as commands. The drone used was again the AR Parrot drone, so the authors were also forced to use the ground station, to which the LEAP motion controller was connected. Therefore the interaction was relayed via the PC instead of a direct interaction with the drone.

Such approaches are gaining traction, but instead of using expensive proprietary motion controllers, the authors are rather using specialized devices such as gloves that users wear, equipped with motion sensors. These approaches are easier to develop, so there is no lack of ”do it yourself” projects such as [3]. Such an approach is very intuitive to the user and requires no camera and no computer vision processing, it simply requires a communication between the device and the drone. However, this does not allow the user to use the hand that is used for controlling the drone for anything else and the control itself is quite difficult to master precisely.

A similar approach is a so called Maestro Glove [21]. The approach is very similar but is no longer a ”do it yourself” project, it is a completed product being sold by a company. Since we are in an era where smart watches are popular devices, which are equipped with motion sensors, it was only a matter of time for such an approach to be miniaturized into a simple application running on an Apple Watch, as can be seen here [4].

There do exist some gesture recognition systems that did run on a drone itself. However they still required expensive RGB-D sensors to be mounted on the drone, which still comes with a big drawback. RGB-D sensors in general are not small and light devices, they are quite the opposite. That means that drones that carried them were required to have a very big upper limitation on the payload that they could carry. And then there was still a problem with computational power, which was solved by using a full-fledged PC on the drone itself. This also meant more payload and with it, a bigger and a more expensive drone.

Researchers in [5] created an implementation of gesture control using an inexpensive drone, the Parrot AR and it's on-board RGB camera, which is similar to what we wish to achieve. The system was based on face tracking and hand gestures. The drone would track the operators face and orient towards it. Then it would detect users hands in relation to the tracked face. They then use the pose of the face as the angle between the human and the robot's point of view. They use the orientation of the hand direction with respect to the location of the face, which is interpreted by the drone as a directional command. When it reaches the optimal position the drone stops. The system has many drawbacks however. Since the Parrot AR drone is lacking an on-board computer, everything needed to be processed on an off-line Linux PC. The user would also have to wear colored gloves, so that the drone would be able to detect and track them. It also limited drone's movements in a very restricted space near the user.

One example of gesture control where, in theory, the UAV is not limited to the area near the user is [22], where authors wanted to track and recognize gestures for signaling aircraft. For that purpose they created a dataset of videos with NATOPS aircraft handling signals. The contribution of this work is also a unified framework for body and hand tracking. Our work was partially inspired by this framework, but there are major differences. Authors of [22] use a 3D camera that is stationed on the ground, much like in [1]. However we aim to implement a similar framework directly on the UAV with only a monochrome camera, which requires very different underlying methods..

Described approaches and research are good first steps towards controlling drones using gestures. However with the development of more powerful hardware, in terms of computers and smaller and more accessible drones, we can advance above systems. One problem is of course the need for external devices, because the user does not, in that way, really interact with the drone itself. It might be a good approach in some cases, where the drones are not in line-of-sight of the user. But in this work we would like to gesture control drones that are in line-of-sight.

The payload of lighter, smaller and more affordable drones is also a big issue, and so heavy 3D sensors are also a problem. For a user wearing special gloves may also not be practical and undesirable. Therefore the system should not require any external devices. Since the more affordable drones come with mobile processors (that include GPUs) and a rather simple RGB camera, we would like to focus on such hardware. For these reasons we need to reduce computational complexity to such a level where it can be run on a mobile GPU. We solve this by limiting the video frames that are processed through action detection, which will be described in the following subsection.

## 2.3 Action Detection

The task of human action detection has the goal of localizing a human action in a certain time frame within a video and it has many applications such as sports video analysis, human to robot interaction and many more. Area of research that tries to detect actions is very related to gesture recognition and there was much research done, where authors would detect actions in the scene and try to label them (with recognized actions). This might refer to either the position in the video where the action is happening or in which part of the video the action is happening.

However there was little research done on performing action detection online, which is the scenario where we have to detect actions on always incoming video instead of capturing the video, storing it and processing it after the events have already occurred. In the best case such action detection runs in real time. One of the reasons for lack of research in this area is lack of labeled data, which is required for learning, comparison and evaluation of different methods. It is also a very difficult problem to solve, since actions can occur at any given time.

For our purpose, detecting when the action occurs is very important, since it allows for reduction of computational intensity of the whole system since it enables us to only process certain parts of the incoming video instead of every incoming frame. One could think of it as a synonym for search space reduction.

One example [6], where authors try to push the research of online action detection further, contributes to this problem in three ways. They conclude that to date, no realistic benchmark dataset focusing on this problem has been released. And so they introduce a labeled dataset for the purpose of algorithm evaluation and they collect well performing methods up to the present and evaluate them and compare them. Their unfortunate conclusion is that none of the methods provide a good solution. They also design an evaluation system for online action detection methods and with it evaluate difficult examples such as occlusions and variation in viewpoint. Unfortunately they do not provide (as appears to be the case in most works in this area) time performance measurements. Authors go on to compare three state of the art (and lately very popular) methods for solving the problem, namely Fisher vectors with improved trajectories [23], a deep ConvNet [24] and a Long short-term memory network [25]. They found that all of the above methods struggled with detecting action in a real-time setting. However in general they found that Fischer vectors are better than LSTM, which is better than CNNs. LSTM can use information in a temporal order, which is not the same as having real motion information, but improves the results nevertheless. That said there is still a lot of research pushing CNNs into this area because of their recent popularity.

The latest methods for action detection focus mostly on localizing actions in video in terms of position, using convolutional neural networks. One of such methods is Action Tubes [7], which combines action detection and action recognition. They use motion saliency to find out which regions are most likely to contain an action and eliminate the regions where there are none. The predictions are linked across frames (in time) and this what they call an action tube. There are two separate networks used, a spatial-CNN that operates on static cues and captures the appearance and motion-CNN that captures patterns of movement. Their results are very accurate (achieving 41.2% AUC as compared to leading non-CNN methods, which achieve 22% on the UCF sports dataset). Like previously mentioned methods, they do not provide speed measurements. They do provide an implementation of their method and according to our tests it is nowhere near real-time.

Despite the extreme speed increase that CNNs have achieved in the recent years these approaches are still too slow and too complex to meet our requirements. In fact it is hard to find any information on the time performance of such methods, as they simply disregard the importance of it. Latest research also focuses on improving the accuracy of localization in terms of position in each individual frame instead of temporal position, or position in time. Besides the mentioned drawbacks CNNs also require the GPU to achieve stated performance, which increases the processing power and GPU memory demand, both of which are sparse on the target platform. Therefore we would like a simple method, which is not required to be accurate but should be able to detect when the action is being performed and use as little resources as possible for doing so. It would also be beneficial if the method can run on the CPU.

## 2.4 Pose Estimation and Action Recognition

The task of gesture or action classification differs from action detection in terms of input and output. For action detection, we wish to localize the human action within a video whereas with gesture classification we are required to correctly classify a sequence in which the action was performed. In a way one could say that the output of action detection is the input to gesture classification. The output of gesture classification is the correct annotation of the given time sequence, where an action was performed.

There has been a lot of research on human pose estimation in the recent years as this is one of the most popular research fields in computer vision. Mainly researchers focused on still images instead of videos, which provided some very well performing methods. But video introduces another component, namely the temporal one. One might think that this would be an even easier problem to solve, since consecutive frames are correlated and the pose should not vary frame to frame too much.

So it is natural to ask ourselves, if human action recognition benefits from pose estimation? There was a research published, with the exact title as is the question [8]. In the paper authors examine the importance of high-level features compared to low-level features for pose action recognition. Pose-based approaches stem directly from the definition of an action as a sequence of articulated poses and are the most straightforward to consider. Pose-based features have many advantages. They suffer little of intra-class variances. 3D skeleton poses are viewpoint and appearance invariant or in other words actions vary less from person to person. Using pose greatly simplifies the learning for the action recognition itself, since the relevant high-level information is already extracted. Of course, appearance based features also have advantages. There is almost no high-level processing and they can bypass the difficulties of pose estimation and features are not restricted to the human body! They are applicable where pose estimation is difficult, for example with monocular views (on RGB cameras) or on very low resolution input. They go on to perform a series of tests comparing pose-based features with appearance based features. Authors use the same action classifier for both sets of features, which is a Hough transform-based voting framework for action recognition, presented in [26].

They use relational pose features describing geometric relations between specific joints in a single pose or a short sequence of poses. Relational pose features were introduced in [9] and have been used for indexing and retrieval of motion capture data. Authors [26] conclude that pose-based features out-performed appearance features by 7-10% and that combined features don't perform much better but still better than using appearance features alone. They also performed Gaussian noise tests to see how it affects classification. Plane features (a plane spanned by three different joints and distance from joints to the plane) are not affected much - they degrade at about 75 mm of Gaussian noise added. There is redundancy in pose-based and appearance based features, when combined. They show that even with high level of noise, the pose-based features either matched or outperformed appearance-based features. This shows that perfect pose estimates are not necessary! Their last important argument in the described work is that appearance based features are good when pose is hard to extract. So a combination of appearance and pose based features would be ideal despite reaching a lower classification accuracy on their particular dataset.

Similar results were achieved by [10], where authors set out to compare low to high level features. They found themselves short of a dataset with accurate ground truth for pose estimation and decided to create their own, where each frame in the dataset is annotated using a 2D articulated human puppet model that provides scale, pose, segmentation, coarse viewpoint and dense optical flow for the humans in action. They refer to this dataset as J-HMDB. Since the dataset is not too large and is very well annotated, we use it to evaluate the system as well. For their evaluations they used the dense trajectories algorithm [27]. They compared low level features (dense optical flow), mid-level features (bounding boxes of people) combined with low-level features and high-level features (pose estimation). They found that high-level features greatly outperform low-to-mid level features. Or in other words, they find that using pose estimation is critical in improving robustness and accuracy of action recognition.

Work by [28] in 2013 pushed the research on pose estimation further. Their important insight was that training SVMs for individual joints carries insufficient discriminating information. Instead, body parts are used as building blocks which is more meaningful and compact. And as an alternative line of work at that time of what researchers did in the past, they use features defined as spatial (joint configurations) and temporal (set sequences). This in effect defines actions as sequences of poses in time, where poses are spatial configurations of body joints, which is supposed to be the way that humans understand actions. Besides this they note that in pose estimation datasets variation is huge (for example in environments where action is taking place) and that the occlusions are extremely hard problems to solve. They implement the pose estimation as groups of five body parts, which will represent action. They declare their spatial domain with co-occurring spatial configurations (poses, spatial-part-sets). Their temporal domain are the distinctive co-occurring pose sequences or temporal-part-sets. They assume that groups have related motion given a specific action. Parts are then detected using [29], which was at that time the state of art for pose estimation. Part sets are detected in videos and each video is then represented as a histogram of detected part-sets. Histograms are then classified as actions using support vector machines.

Given the recent popularity of convolutional neural networks it was only a matter of time that researchers would apply them to pose estimation. One paper that pioneered the CNNs in this field of research are Flowing ConvNets [11], which were applied to human pose estimation in videos. Authors combine convolutional neural networks with temporal information through multiple frames using optical flow. Their key contribution is in exploiting the temporal information in videos. They first use the convolutional neural network to regress the joints and then use dense optical flow to warp the coordinates of joints onto the next target frame which effectively propagates the pose estimation through the video. They also use a CNN with additional layers that they call spatial fusion layers and are able to learn an implicit spatial model of human pose layout. These layers allow them to remove pose estimation errors that are kinematically impossible. Their method outperformed traditional pose estimation methods, such as pictorial structure models [30], poselets [12] and random forests methods. They also achieved better results than other researches that tried to use simpler CNN models for pose estimation such as [31], where optical flow was simply used as an input motion feature directly to the CNN.

Using CNN for for pose based action recognition continued to be a trend after the promising results of [11]. In a recent work authors introduce a Pose-based CNN descriptor [32] (P-CNN) for action recognition. Provided with body joints over time, their descriptor combines motion and appearance features for body parts. They state that the reason for a new descriptor based approach is in the disadvantage of global approaches, which may not be optimal in recognizing fine actions such as distinction between correct and incorrect golf swings. They believe action recognition can benefit from the spatial and temporal detection and alignment of human poses in videos. As Fisher vectors with dense trajectories established themselves as a state-of-the-art action detection method, they use the dense trajectory features in combination with their method. In short, authors take body joints and split them into parts (right hand, left hand, upper body, full body, full image) then they compute optical flow for each of them. Poses are extracted for individual frames using the state of the art pose estimation method by Yang and Ramanan [29]. Computed optical flow is then input into RGB CNN and Flow CNN for each part for each frame. Then both are combined into P-CNN, the final feature. Finally the outputs of P-CNN are classified using a linear SVM to recognize gestures or actions. They evaluate their results on the two datasets JHMDB and MPII Cooking Activities. There are some valuable insights in their work. They note that combination of parts improves performance and that optical flow descriptors outperform appearance descriptors and both of them combined increase the performance further. They found that their P-CNN descriptor is very good at describing fine-grained actions. Finally they argue that correct estimation of human pose leads to significant improvements in action recognition. That implies that pose estimation is a crucial part in action recognition. As was the case with previous methods, the P-CNNs are slow. They suffer the inherent computational complexity of Ramannan and inference through two separate CNN-s (RGB and Flow CNN).

The authors did not state the time measurements but in our test we were able to achieve 1 minute and 15 seconds for extraction of OF features alone in a 30 second video. So it is clear that the method is not suitable for near real-time use. The biggest pitfall of CNN methods was the computation of pose that is used for features in such approaches. If human pose estimation was calculated fast enough using CNNs, then it would not be necessary to stack above it another CNN for action recognition, where more traditional models were accurate in making the final prediction just as good and faster. In that way one could look at the computed pose by CNN as an input into gesture classification.

Human pose estimation received a lot of attention lately with most of the new methods proposed using CNNs. With promising accuracy of achieved results however, there was still an issue with speed. The method of [11] was rather slow. Speed and accuracy of such methods was then greatly improved with research done by [13]. Their sequential prediction framework for learning rich implicit models called Convolutional Pose Machines is a sequential architecture composed of neural networks. Instead of explicitly parsing belief maps that are output from a CNN, or post-processing the output like it was done so far, they train neural networks to operate directly on belief maps. This allows them to propagate a belief map through multiple stages. In other words, they refine the belief map with multiple networks, where the map is being passed through them as an output of the previous stage and input into the next one. With this method they had greatly outperformed state of the art methods on datasets MPII (10% better results than second best performing method) and FLIC (12% better performance than second best performing method). CPMs, despite its speed improvements over competitive methods, still did not run in real-time.

Following the multi-stage approach of CPMs authors [14] developed the first real-time multi-person pose estimation system. Their method differs from most of the state-of-the-art methods in that they consider the whole system, including a person detector, in order to support multiple people, which provides the algorithm with locations and scales of persons in the target image. They use a two-branch multi-stage CNN (VGG-19). The first input into the network is a color image. First stage of the network then computes a confidence map of joint locations and an affinity field that encodes part-to-part association. The confidence maps and part affinity fields are then forwarded into the next stages, which refine it, with intermediate supervision at each stage. The parallel inference allows for a huge speed increase. The method out-performs rival methods on both MPII and COCO datasets. The biggest improvement however lies in speed, as this method is capable of running at 8.8 frames per second for a video of 19 people on a mobile NVIDIA GeForce GTX-1080 GPU. As this method satisfies the real-time requirements and accuracy of our system implementation, it will be explored in detail in the following chapters.

Based on the status of state-of-art research in action detection, human pose estimation and action recognition, we believe it may now be possible to combine the three into a real-time system for gesture control.

# Chapter 3

# Mobile platform for gesture control

## 3.1 Assembling a quadcopter

So far we have learned that simple, lightweight drones such as the *AR Drone*, currently, do not meet the requirements for complex system implementation. The reason is mainly in two parameters. One of them is *computational power* of the drone platform. If we wish to implement an online system that is running in real time or at least near real time, a drone needs to be able to process the complex algorithms of our system. This is not an easy task, since we have learned that human pose estimation can be achieved in real time only when using advanced GPUs, such as a mobile *NVIDIA GeForce 1080 GTX*.

The other big limitation is *payload*. By definition payload is the added weight with which the drone can still fly safely. This implies that the drone must have enough payload capacity to carry the necessary components that provide computational power - its on-board computer, and with it a capable RGB camera, with which we capture the user while performing gestures. This is not trivial, since capability to carry more payload is achieved through engine power, which are in terms supported by other components such as the size of propellers used, the power of electronic speed control units and last but not least they influence battery life heavily. Achieving higher payload thus means upgrading to more powerful engines, bigger propellers, a bigger battery and so on.

These two limitations are also the reason for current approaches using offline PCs that process algorithms and the use of external devices, not mounted on the drone, such as Microsoft Kinect and other 3D sensors, which are simply too heavy to be mounted on smaller drones. And so, we must find a way to provide enough computational power, while keeping the payload requirements low, which allows the use of smaller and cheaper drones.

### 3.1.1 The frame

Selecting a frame on which all components will be mounted seems like a very hard choice. In reality however, it is not, unless you care very much for the visual appearance of the assembled platform. It is important thought, to try to figure out how all components can be mounted on it based on size. Another important factor in this decision is the sturdiness or firmness of a frame, which needs to be good enough to support the weight of the central part, where battery is usually located. Frames can be made out of plastic, carbon fiber, aluminum and so on.

The best option in this case is carbon fiber but it is also very expensive. The next best option is sturdy plastic. There are a few manufacturers of do-it-yourself kits, that provide a frame and engines combinations. It is a safe bet to find a kit that already combines both, since the manufacturer will make sure that the frame can support the tension that is imposed on the frame while flying and that the engines are strong enough to provide enough lift force for flying. We took a slightly special approach with our frame selection and decided to use spare parts that are sold for the *Parrot Bebop 2* drone. We used the *Parrot Bebop 2* frame, shown on Figure 3.2, and propellers combined with *Parrot Bebop 1* engines, shown on Figure 3.1.

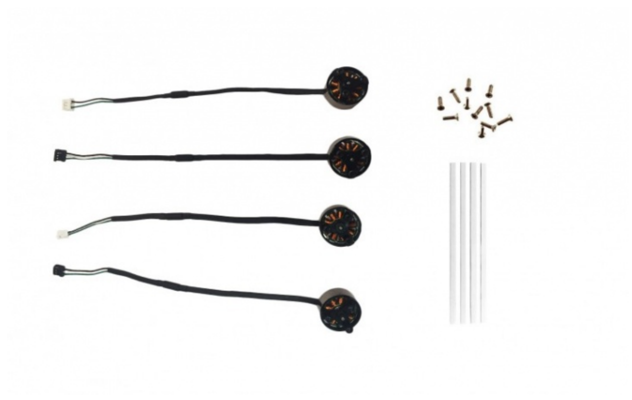


**Figure 3.1:** An image of Bebop 1 engine kit sold by Parrot. Image source: parrot.com

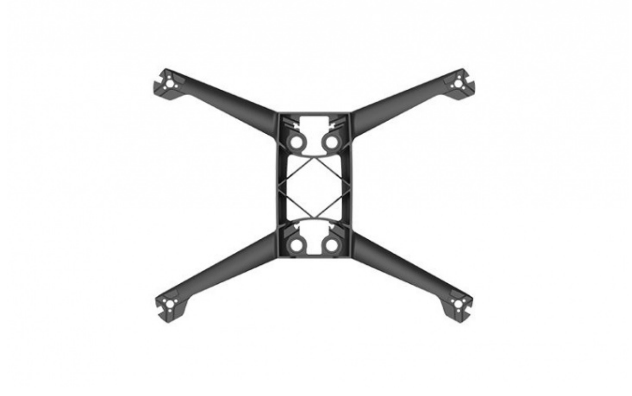


**Figure 3.2:** An image of Bebop 2 frame kit sold by Parrot. Image source: parrot.com

We chose the frame from Bebop 2 because it is bigger and gives us more space and we used Bebop 1 engines because they offer enough power, have less drain on the battery and reduce stress on the frame. Another reason for this choice is the fact that we wish to prove, that the required platform does not have to be big and expensive. Because of its size it is better suited for indoor flights, as it does not require much space but it has no issues flying outside.

The width of the frame is 250 millimeters, its length is 200 millimeters and its height is just 20 mm. The weight of the frame without electronics is therefore only 43 grams. It is a sturdy frame made out of hard plastic. We added 3D printed landing feet as a small modification to the frame, to provide more ground clearance and a more stable, non-painful landing. This way the drone is 6 centimeters above ground and all its electronics are safe. It also allows us to mount the battery below the frame. The frame is also upgraded with a custom 3D printed mounting plate for the on-board PC, electronic speed controller and a front camera. Below the frame we added a custom 3D printed battery mounting plate, to fit and secure the battery in place while flying.

### 3.1.2 Flight Management Unit

Selecting a flight management unit (*FMU*) or an autopilot is the most important part of building a quadcopter. The FMU provides the necessary low-level algorithms for stabilizing and flying the drone and serves as an interface for flying. It enables flight using either autopilot mode via GPS controlled way point flying or remote controlled flight, using an RC transmitter and receiver combination. There are not many FMUs to choose from in the research world. There are proprietary FMUs available, from companies like DJI and Parrot but they are not open source, to tinker with. They are instead meant to be an out-of-the-box solution, used with the company's software.

The *PixHawk PX4* flight stack is an open-source and open-hardware project [33] that aims to provide a state-of-the-art autopilot hardware and software in a single package. Because of its open-source initiative, it is very suitable for researchers. The *PixHawk PX4 FMU* used to be the reference hardware implementation for the PX4 flight stack. However due to its popularity it was mostly sold out and later improved on by other manufacturers, who released hardware capable of running a PX4 flight stack.

Since the PX4 flight controller wasn't available, we decided to use *Holybro PixFalcon* flight controller [34]. It is a derivative of the PX4 design with improved features but with less I/O capability, to reduce size and save on weight. It comes with a Cortex M4F on-board processor, with 256 KB of SRAM and an additional fail safe System-On-Chip (*SoC*) ARM Cortex M3. Fail-safe SoC enables in-flight recovery and manual override in case anything goes wrong. It is equipped with an SD card slot and offers 8 PWM ports for engine control. It also has a magnetometer, a gyroscope, barometric pressure sensor and an accelerometer. A *PixFalcon* is shown on Figure 3.3.

The module offers flight support for any multi-copter, rover or a boat. It comes with pre-loaded frame settings (a similar frame to Bebop 2 frame is already supported) and also allows users to specify its own frame characteristics.

The FMU directly controls the Electronic Speed Controller (*ESC*), which distributes power to the engines. To save weight, we decided to use a 4-in-1 ESC unit. The ESC unit is therefore a single board to which the engines are soldered onto instead of having 4 ESCs, one for each engine. We used the *Afro Race Spec 20A 4-in1 ESC*.

To interface with the *PixFalcon* hardware, *QGroundControl* [35] (*QGC*) open-source software was used. It provides full flight control and mission planning for any *MAVLink*[1] [36] enabled drone. This software also enables

[1]MAVLink is a Micro Air Vehicle messaging library for lightweight communication between drones and ground stations.

modifying various settings of the autopilot, from sensor calibration, battery settings, frame configurations to the actual autopilot gains that have a direct impact on the autopilot behavior.

**Figure 3.3:** An image of the PixFalcon FMU. Image source: holybro.com

**GPS**

We use the *Ublox Neo-M8N* GPS module with integrated Compass. The GPS module is capable of position tracking via GPS, GLONASS, Galileo and BeiDou positioning systems. It is a standard precision GPS with a low cold start of 28 seconds.

### 3.1.3 Dedicated on-board computer

Choosing an on-board computer is not an easy task. It should be able to handle the vast amounts of processing that is required by computer vision algorithms. This is not a small requirement, since even full-fledged desktop PCs struggle to perform this in real-time. In the last couple of years there has been a lot of innovation in miniaturizing hardware components that bring a tremendous amount of computational power with them. It started with computers such as Raspberry Pi and quickly progressed with the availability of ever more powerful ARM CPU and GPU combinations, that were pushed ahead with smartphone development. Very recently it has been announced that smartphones are now capable of immense computational power that in some cases outperforms desktop-class CPUs as well as supporting deep learning inference. The same processors that are found in smartphones are also available on small compact PCs. There are many such PCs out there, but we focused our decision among three - *Odroid XU4*, *NVIDIA Jetson TK1* and *Snapdragon Flight*. We will briefly review the three choices and explain our difficult but necessary choice.

The *Odroid XU4* [37] is a small computer on a single board. It features a strong quad-core ARM A-15 processor with each core running at 2.0 GHz and a quad-core A7 processor with each running at 1.4 GHz. It has 2GB of LPDDR3 memory and comes with a lot of input and output ports, namely 2 USB 3.0 ports and one USB 2.0 port, an Ethernet port, and HDMI for graphical output. It uses an SD micro card or an eMMC as a hard drive. Its advantage is that it is powerful enough to run a full Linux distribution, while still being small and energy efficient and reasonably priced. It also comes with a separate ARM Mali T-628 GPU. XU4 is shown on Figure 3.4, with its components listed.

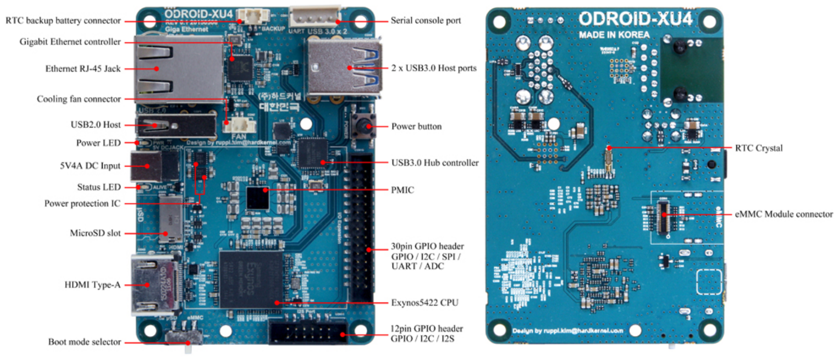


**Figure 3.4:** An image of Odroid XU4. Image source: hardkernel.com

The *NVIDIA Jetson TK1* [38] is an embedded powerhouse. It features a quad-core ARM A-15 processor with 2 GB of RAM. Its obvious advantage is the *Tegra K1* GPU with 192 CUDA cores, which is powerful enough for real-time neural network inference. NVIDIA even offers a Tegra version of OpenCV (the most popular open-source computer vision library) for faster development and deployment of cutting edge applications. The chip itself comes in a very lightweight and small factor. However, it has a big drawback of not having any I/O by itself. For I/O an additional board has to be used, on which the chip is mounted. Unfortunately *NVIDIA Jetson TK1* was not available at the time of development. Another issue is the socket, since it requires a special board that hosts the Jetson and so mounting on a drone would be quite an adventure, especially because these boards would be released well after the Jetson itself. *NVIDIA Jetson TK1* embedded development board is shown on Figure 3.5.

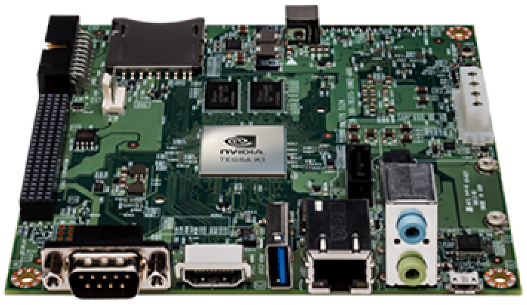


**Figure 3.5:** NVIDIA Jetson TK1 embedded developer kit. Image source: nvidia.com

The *Qualcomm Snapdragon Flight* [39] module has been introduced as an all-in-one solution, which could easily be the first choice. *Snapdragon Flight* combines a 4K front camera, optical flow camera, GPS, Wi-Fi connectivity, high computational performance and a flight management unit! As the other two competitors, it features 2 GB of LPDDR 3 RAM, with a Snapdragon 801 ARM quad-core processor. The clear benefit of *Snapdragon Flight* is that it is an all-in-one, which makes it very easily mountable on the drone. However, this would defeat the purpose of building a cheaper drone, since *Snapdragon Flight* is very expensive. Another drawback was its early life. Support for *Snapdragon Flight* was not very good at the time of development. It also had issues with drivers for its embedded components, so it was very difficult to work with. To add to the drawbacks, there was also an issue with its cooperation with 3rd party hardware. Therefore, we dismissed the *Snapdragon Flight* (shown on Figure 3.6) as our choice.

**Figure 3.6:** An image of the Snapdragon Flight. Image source: intrinsyc.com

Because of the drawbacks presented with *NVIDIA Jetson TK 1* (especially availability in this case) and *Qualcomm Snapdragon Flight*, we have decided to use *Odroid XU 4* as a dedicated on-board computer. This was a difficult choice, since deep learning applications are very specific about the choice of GPUs on which they run. We will explain later how we solve this problem and discuss what choices there are for the near future.

### 3.1.4 On-board RGB camera

To capture the RGB video stream we use an *Odroid USB-CAM 720P*, capable of delivering 720p HD resolution video in a 16:9 aspect ratio. It has a 1 MP CMOS sensor. The camera board can deliver video at up to 30 frames per second. We chose this camera because of its form factor (when stripped down to its board by removing the enclosure, the camera takes very little space on the drone) and its compatibility with the *Odroid XU 4* on-board computer.

### 3.1.5 A transmitter and a receiver

Transmitters allow us to send commands via radio transmission to the receiver that is mounted on the controlled platform. There are not many restrictions in selecting a transmitter and a corresponding receiver. However, since we chose a *PixFalcon* for an autopilot, we need to use an R/C combination that is supported. We would like to have manual control over the drone as well as automated control. Therefore we would like to have a combination that supports at least 7 channels and PPM (Pulse Position Modulation), an output for analogue signal that uses a single wire for a stacked signal - instead of a separate wire for each channel, which makes it easier to connect to the autopilot. Because of our particular choice of PixFalcon, we are able to choose between *Futaba S.BUS* or *Spektrum DSM/2/X* pairs, since they are officially supported. We decided to use *Spektrum DX7*, a 7-channel DSMX telemetry system remote controller (transmitter) and *Spektrum AR7700* receiver, which supports 7 channels and PPM output.

In our case manual control is used as a backup scenario, if in any case a need would arise for manual override of automated flight. Otherwise the commands are sent to the autopilot directly via the on-board computer, which eliminates the need for a transmitter and a receiver. It is also great fun to be able to fly a drone manually, while it is not performing complex tasks.

### 3.1.6 Assembling it all together

Assembling the drone is a bit of an art, since the components need to be attached in a safe manner to the body. For this purpose we have designed some mounting plates by ourselves and 3D printed them.

Interfacing with the autopilot via *QGroundControl* is very straightforward and it doesn't take long to set up the remote controller and calibrate all the

| Quadcopter parts | |
|---|---|
| **Component** | **Model name** |
| Frame | Bebop 2 frame kit |
| Engines | Bebop 1 engine kit |
| ESC | Afro Race Spec 20A 4-in1 |
| Propellers | Bebop 2 propeller kit |
| FMU | Pixfalcon |
| GPS Module | Ublox Neo-M8N |
| On-board computer | Odroid XU 4 |
| On-board camera | Odroid USB-CAM 720P |
| Transmitter | Spektrum DX 7 |
| Receiver | Spektrum AR7700 |

**Table 3.1:** A summary of quadcopter parts chosen to be used for our mobile platform.

sensors. *Odroid XU 4* has been equipped with an SD card, with installed Linux Ubuntu 16.04 LTS, *OpenCV* and *Robot Operating System* (ROS).

Putting all of these components together results in an excellent small drone platform with an open source autopilot, suitable for research. It also has an on-board computer with enough processing power for our complex computer vision algorithms. A fully assembled drone is shown on Figures 3.7, 3.8, 3.9 and 3.10. In manual flight tests, the drone is quite stable in indoor flights, but due to high speed and maneuverability it still requires some invested time from the user, to acquire the skill of flying it safely and efficiently. Safety is not to be overlooked when flying, as the engines for the propellers are quite strong and the propellers are not protected in any way. All parts that make up the quadcopter are listed in the Table 3.1 for a better overview.

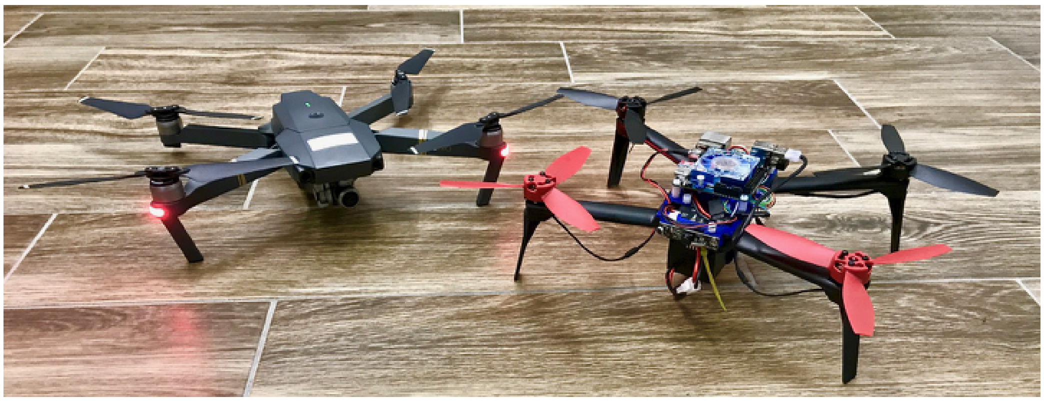


**Figure 3.7:** The custom built drone next to a very popular DJI Mavic Pro drone, to get a better comparison of size.

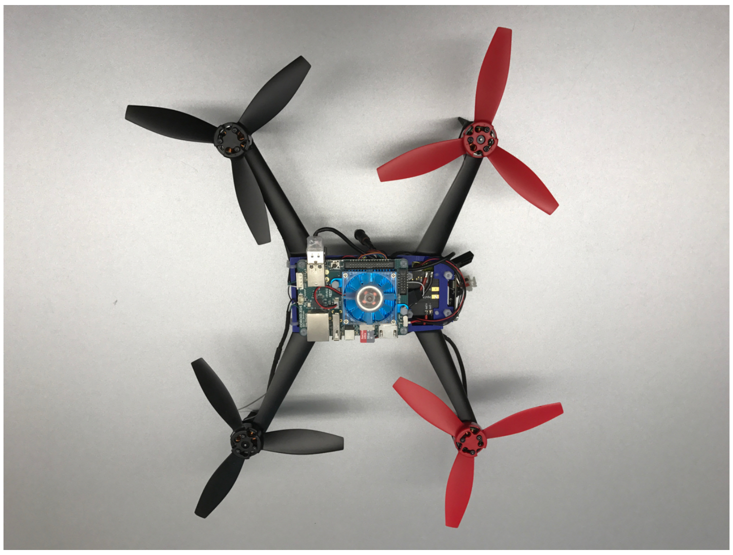


**Figure 3.8:** Top-view image of the assembled drone.

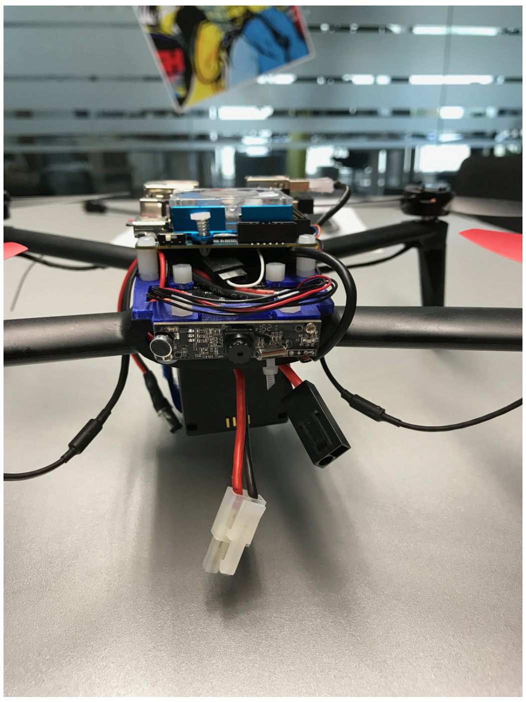


**Figure 3.9:** Front-view image of the assembled drone.

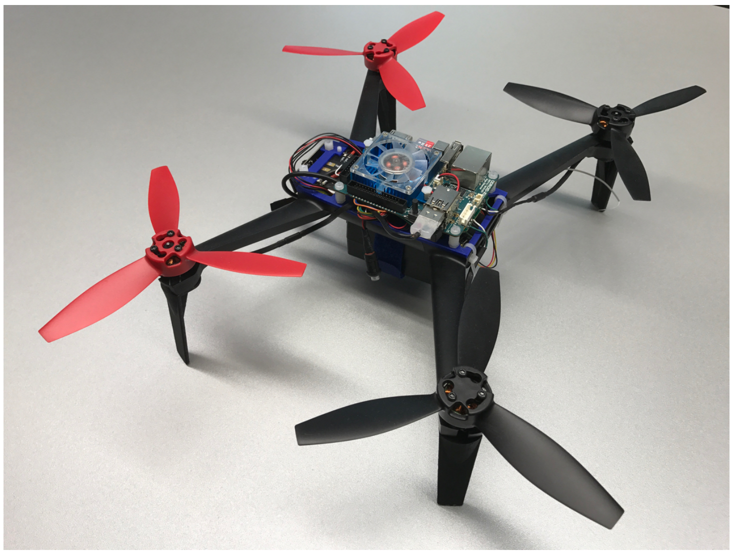


**Figure 3.10:** Side-view image of the assembled drone.

# Chapter 4

# Computer vision methods for gesture control

## 4.1 System description

We developed a gesture control system for lightweight small UAVs. Our focus was on quadcopter drones but it can be used widely on different platforms. Due to our choice of platform, we focused on developing a gesture control system, that is efficient. We did not want to have a system with complex methods, because we wanted to be able to use gesture control in real-time, while executing it on the drone itself. As we have seen in the previous chapter, where we describe the status of computational performance on small platforms, this is not an easy task. And sure enough there is a part of the system that requires immense computational power. We will now overview the whole system and explain its components in detail.

The gesture control system is composed of three main components. The first important component of the system is *action detection*. We use action detection in order to improve computational efficiency of the whole system. The second component is *pose estimation*. With pose estimation we are able to obtain important features that are used in the final component.

After computing features we use *gesture classification* to infer the performed gesture. This final component provides the command to the drone.

In the system action detection is very important. If we were to simply estimate the person's pose in order to get distinct features, from which we can interpret actions, we would not be able to achieve real-time performance or even come close on limited processing power that we can get on reasonable hardware. Due to the latest advancements in the field of pose estimation, we hope real-time performance will soon be achievable without this component. Currently we are able to achieve near real-time performance running pose estimation system on a powerful PC. Therefore we need to detect when the action or gesture is taking place in a video sequence so that we can focus the processing power on a particular segment. With this component we are able to reduce the requirement of processing 30 frames per second (which is the usual frame rate of video capturing) to only a few limited frames in the whole sequence.

After we narrow the sequence down to a specific segment, we are able to use complex pose estimation algorithms that provide us with important features. The result of pose estimation is a set of joint positions as coordinates in an image. As we have seen in [8], pose features are a great benefit to action recognition and they outperform low-level features. From [26] we have also learned that joint localization does not have to be perfect to still provide quality features. We have chosen to use the state of the art method for pose estimation [14] for the gesture control system. This method is based on deep convolutional neural networks (CNNs). This choice obliges us to use a specific powerful GPU, that is not available on the drone. We mentioned in the previous chapter, that the drone is equipped with a GPU, which has a lot of compute performance. Unfortunately deep learning frameworks that are required to use CNN inference do not support such GPUs. In fact most (if not all) deep learning frameworks currently support only NVIDIA GPUs that have CUDA cores. This means that a part of the system still needs to be processed on an external PC, which is equipped with such a GPU.

We will describe in a later chapter why this is not a big issue and how the system is still ready for real-time and online deployment. Gesture classification has been implemented on top of pose features in a bag of words approach. We learn gestures as words and then try to match a newly detected action to these words using an SVM classifier. In the following subsections we describe in more detail each component of the gesture control system and its underlying methods.

A system overview diagram is shown on Figure 4.1.

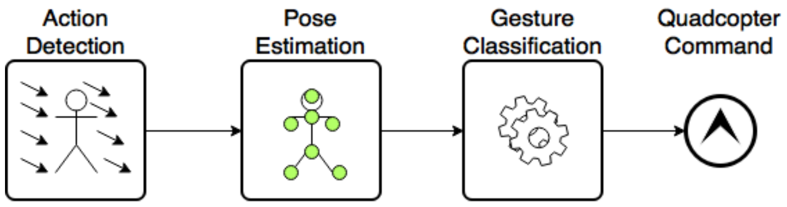


**Figure 4.1:** System overview, showing three modules of the gesture control system. It starts with action detection, which forwards the frames on which action was detected to the pose estimation algorithm. After the pose is estimated, joint positions are forwarded to the gesture classification algorithm and its output is translated into the final command for the drone.

## 4.2 Action Detection with person tracking

Action detection itself is a big task to tackle. Therefore it is split into three sub modules. We first detect the person with a *person detector*, which provides initialization coordinates for a short-term tracker that keeps track of a person through subsequent frames. Since tracking is an extremely hard task, we make it easier by re-detecting a person every number of frames. The reason for not using person detection in each frame is speed. A combination of a fast tracker and a person detection is much faster. While making sure that we are focused on a person, *optical flow* is employed as an indicator of action. When there is enough optical flow generated an action is considered to be in progress. Optical flow is also produced by movements of a drone so we have to subtract that from the optical flow that is generated by a person. A flow chart of action detection with its sub-modules is shown on Figure 4.2.

To increase the processing speed of the system we also decreased the frames per second provided by the camera. Instead of 30 FPS as is usual we use 15 FPS. Person re-detection is done every 30th frame or every 2 seconds. We experimentally determined this number to be sufficient and a good trade-off for speed.

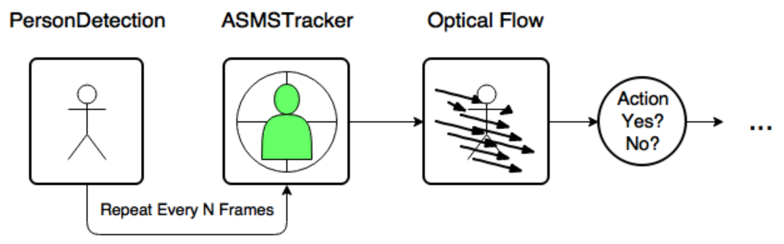


**Figure 4.2:** Overview of the action detection sub modules. Action detection begins by detecting a person accurately, tracking it through several frames and calculating optical flow that is later filtered and used to determine if action is in progress or not. Person detection re-initializes the ASMS tracker every N frames to prevent tracking error from growing.

### 4.2.1 Person Detection

There are many person detectors available, but we must be careful, since not many are fast. Most detectors focus on accuracy rather than speed. They also wish to handle a majority of cases, eg. when people are facing the camera sideways or are covering one another. For our use case we do not need to handle these cases and we assume that a person is reasonably aligned with the drone. In a research study done by [40], authors analyzed how fast state of the art person detection actually is. The results were surprisingly bad. For example [41] claimed to perform at 100 FPS but did not achieve such results in realistic test scenarios. It was also implemented on a GPU, which would require the system to dedicate more scarce resources to this task. A similar performance is achieved by [42], which is a part of open source computer vision library OpenCV. We decided to use this person detector for the system. It is implemented on CPU and GPU, with GPU implementation of course being faster. We dismissed the use of GPU and decided to use the CPU version to support our vision of a on-line system, where GPU is dedicated solely to pose estimation. This sacrifice requires us to use person detection less often but this is not a big issue, since the tracker is able to keep up for a good amount of time.

Person detection using histograms of oriented gradients is a fast machine learning method using well-normalized local histograms of image gradient orientations in a dense grid. The resulting descriptors are then classified using a linear SVM. They rely on the distribution of local intensity gradients to describe the object appearance and shape.

**Histograms of oriented gradients**

The method is rather simple and works as follows. The image window is first split into cells forming a dense grid. For each cell a 1-D histogram of gradient directions or edge orientations is computed. Magnitude of gradient is computed according to

$$|G| = \sqrt{I_X^2 + I_Y^2} \tag{4.1}$$

where $I_X$ and $I_Y$ are $x$ and $y$ image derivatives. The orientation of the gradient is computed as

$$\phi = \arctan \frac{I_Y}{I_X} \quad . \tag{4.2}$$

Each pixel within a cell has a weighted contribution based on the values of the gradient magnitude and orientation. Histogram bins are spread evenly from 0 to 360 degrees (if the gradient is "unsigned", otherwise 0 to 180 degrees for "signed gradients"), representing orientations. This is called orientation binning. To prevent issues with illumination they are contrast-normalized by using the surrounding cells merged into a block describing a larger spatial region. This normalized block is referred to as Histogram of Oriented Gradient (HOG) descriptor. We have a choice of using two block geometries, rectangular R-HOG and circular C-HOG. These blocks overlap, so each cell has more than one final contribution in the descriptor. Block normalization is then computed with either L2-norm, L1-norm or L1-sqrt. We used the L2-norm. If we let $\mathbf{v}$ be the non-normalized vector of histograms in a block, then $\|\mathbf{v}\|_k$ for $k = 1, 2$ is its $k$-norm and $c$ is a small constant that does not influence the result, the L2-norm is computed as

$$f = \frac{\mathbf{v}}{\sqrt{\|\mathbf{v}\|_2^2 + c^2}} \quad .$$

The detection window is moved over the image at all positions and scales combined with a non-maximum suppression. Overlapping HOGs are then collected over the detection window and sent to a linear SVM for classification.

Authors of the method describe various design choices of implementing their method for the purpose of human detection and conclude that fine-scale gradients, fine orientation binning, relatively coarse spatial binning and high-quality local contrast normalization in overlapping descriptor blocks are key in obtaining good results. For their detailed discussion an interested reader can refer to the original work [42].

**Conclusion**

Lately HOG for person detection has been used in many application with the reputation of being relatively fast and robust. We found that using HOG as a person re-detection we are able to localize a person quite precisely in non-cluttered backgrounds despite heavy movements of the drone. Since the algorithm is in general capable of detecting more than one person, it could happen that there is more than one person detected, even if there is only one, non-maxima suppression is used. The output of person detection is the location of the person in the image along with an estimated scale.

### 4.2.2 Person Tracking

As discussed earlier, detecting a person in every frame would be slow, so we switch to person tracking, which is faster. Given a bounding box around a person, as shown on Figure 4.3d, we have the $(x, y)$ coordinates of the person's location in the drones view, as well as an estimated scale of the person, which we can use as an input for a rather simple short-term tracker. We assume that the person, which is to provide gestures as commands, is not obstructed by obstacles. This is an important assumption, since a violation of it would require the use of a more complex, and perhaps a long-term, tracker. Our assumption allows us to use a simple but fast tracker based on mean-shift [43], named Adaptive scale mean shift or ASMS. We chose this particular tracker because it is extremely fast and performs well due to its awareness of background appearance and scale adaptation.

**Mean-shift tracking**

In general Mean shift tracking is very simple, it estimates the mean of an underlying probability density function (PDF) and produces a vector as a shift from the current mean to the estimated one. In other words mean-shift is a non-parametric density gradient estimation. It operates in a feature space that can be a color space, scale space and so on. It strongly depends on a choice of a kernel with which we estimate the underlying PDF. This procedure is called kernel density estimation (KDE), which is generally a way of estimating the PDF of a random variable in a non-parametric way. In other words kernels affect the weights that are attributed to each pixel in the frame. Basic mean-shift does not adapt to scale at all, it uses a fixed size window, so its tracking ability is greatly affected if the projection of a person on the frame decreases or increases. For the purpose of tracking we use mean-shift to minimize the distance between two PDFs that are represented by color histograms. Calculating the distance between the two histograms is therefore very important and there are various techniques employed.

In standard mean-shift tracking, as described in [44], a target is represented with a $m$-bin histogram that was estimated with a kernel, positioned at the origin of the selected feature space:

$$\hat{\mathbf{q}} = \{\hat{q}_d\}_{d=1\ldots m} \quad \sum_{d=1}^{m} \hat{q}_d = 1 \quad ,$$

similarly a target candidate is represented with its histogram, positioned at location $y$,

$$\hat{\mathbf{p}} = \{\hat{p_d}(\mathbf{y})\}_{d=1\ldots m} \quad \sum_{d=1}^{m} \hat{p}_d = 1 \quad .$$

We estimate the probability of the feature $d \in \{1, ..., m\}$ by the target histogram according to

$$\hat{q}_d = C \sum_{i=1}^{n} k\left(\|\mathbf{x}_i^*\|^2\right) \delta[b(\mathbf{x}_i^*) - d] \quad . \tag{4.3}$$

Denotation is presented in Table 4.1 for better readability. In this standard mean-shift the target is represented by a unit circle. We use a kernel $k(x)$, which is monotonically decreasing, as well as convex and isotropic. We compute the probability of feature $d = 1...m$ in the target, using the same kernel $k(x)$ with a scaling parameter $h$, according to

$$\hat{p}_d = C_h \sum_{i=1}^{n_h} k\left(\left\|\frac{\mathbf{y} - \mathbf{x}_i}{h}\right\|^2\right) \delta[b(\mathbf{x}_i) - d] \quad . \tag{4.4}$$

Difference between two probability distributions $\hat{\mathbf{q}} = \{\hat{q}_d\}_{d=1\ldots m}$ and $\{\hat{p_d}(\mathbf{y})\}_{d=1\ldots m}$ in ASMS is computed with Hellinger distance of probability

| | |
|---|---|
| $n$ | Number of pixels in the target. |
| $\mathbf{x_i}$ | Pixel locations. |
| $\{\mathbf{x}_i^*\}_{i=1...n}$ | Pixel locations given the origin centered target. |
| $b : R^2 \to 1...m$ | A function mapping values of pixel at location $\mathbf{x}_i$ to index $b(\mathbf{x}_i)$, which points to a bin in the feature space. |
| $C$ and $C_h$ | Normalization constants to ensure $\sum_{d=1}^{m} \hat{q}_d = 1$. |
| $\delta$ | Kronecker delta. |
| $h$ | Scale parameter for the kernel. |
| $\{\mathbf{x}_i^*\}_{i=1...n_h}$ | Pixel locations in current frame, with the target located at $\mathbf{y}$ |
| $n_h$ | Number of pixels in target of current frame. |
| $g(x) = -k'(x)$ | Derivative of the kernel $k(x)$, which is shown to exist for every $x \geqslant 0$, with exclusion of a finite set of points. |

**Table 4.1:** Denotation used in Equations 4.3 and 4.4.

measures, defined as a metric

$$H(\hat{\mathbf{p}}(\mathbf{y}), \hat{\mathbf{q}}) = \sqrt{1 - \rho[\hat{\mathbf{p}}(\mathbf{y}), \hat{\mathbf{q}}]} \quad , \tag{4.5}$$

where,

$$\rho[\hat{\mathbf{p}}(\mathbf{y}), \hat{\mathbf{q}}] = \sum_{d=1}^{m} \sqrt{\hat{p}_d(\mathbf{y}), \hat{q}} \tag{4.6}$$

is known as a Bhattacharyya coefficient between $\hat{\mathbf{p}}(y)$ and $\hat{\mathbf{q}}$. Maximizing the Bhattacharyya coefficient is equivalent to minimizing Hellinger distance. $\hat{\mathbf{y}}_0$ is the starting location in the previous frame for the target search in the new frame. Gradient ascent is used with a step-size, which is equivalent to the mean-shift method. In search of the target we iteratively move the kernel from $\hat{\mathbf{y}}_0$ to a new location

$$\hat{\mathbf{y}}_1 = \frac{\sum_{i=1}^{n_h} \mathbf{x}_i w_i g\left(\left\|\frac{\mathbf{y}_o - \mathbf{x}_i}{h}\right\|^2\right)}{\sum_{i=1}^{n_h} w_i g\left(\left\|\frac{\mathbf{y}_o - \mathbf{x}_i}{h}\right\|^2\right)} \quad , \tag{4.7}$$

where weights are computed as

$$w_i = \sum_{d=1}^{m} \sqrt{\frac{\hat{q}_d}{\hat{p}_d(\hat{\mathbf{y}}_0)}} \delta[b(\mathbf{x}_i) - d] \quad . \tag{4.8}$$

### ASMS Scale estimation

The real advantage of ASMS is its scale estimation, which is added to the basic mean-shift. We will not describe scale estimation in detail, as an interested reader can refer to the original paper [43]. To add scale estimation to the basic tracker a target is represented with an ellipsoidal region and a restriction is imposed on the kernel $k(x) = 0$ for $x \geqslant 1$. The parameter $h$ defines a scale of the kernel and directly affects the number of non-zero pixels in the target. Now minimizing Hellinger distance means maximizing a function of target probability as well as the scale $h$. In other words, when we are moving the kernel we also change scales which leads to each mean-shift update of location and scale. Furthermore two regularization terms are introduced for scale updates to prevent scale under or over estimation. First term enforces an assumption that the scale does not change dramatically from frame to frame and the second term forces the search window to include some background pixels, in effect, having a slight bias towards the largest scale among those possible. ASMS also uses Backward scale consistency check, which validates the scale estimates from steps $t-1$ to $t$ and $t$ to $t-1$, provided by reverse tracking. This prevents scale implosion in cases where there is background clutter.

### ASMS Background Ratio Weighting

Another important feature of ASMS is Background Ratio Weighting (BRW). It is a ratio maximization instead of Bhattacharyya coefficient maximization. Numerator in the ratio is defined as Bhattacharyya coefficient of the target and the denominator as a Bhattacharyya coefficient of the background. This feature allows ASMS to discriminate the target by exploiting the object neighborhood.

**ASMS Performance**

In terms of performance ASMS was subjected to various evaluations and benchmarks and was found to be comparable with other SOTA methods in 2013. The best results were achieved on sequences with scale changes and small amounts of background clutter. In general ASMS experienced a drop in performance where there was significant background clutter and scale remained the same throughout the sequence, since estimation errors then induce a larger drift. ASMS was also evaluated on Visual Object Tracking (VOT) Challenge [45], where it proved to be robust but held back with accuracy. However, where ASMS lacks in accuracy it makes up with speed, which is perfectly in line with our requirements. Since we perform re-detection, robustness is a plus but not a priority requirement, and accuracy can be average since we only wish to roughly localize the person. Our priority requirement is speed. On average it has a processing speed of 6.1 milliseconds per frame, which is still significantly faster than other SOTA trackers.

**Conclusion**

Person tracking is not an easy task with many possible failure cases where there are occlusions, illumination changes, scale changes, background clutter and so on. In our case sudden movements of the drone can lead to drift due to sudden change of both background and foreground. Therefore the use of person detection is a huge benefit. It allows the tracker to re-detect the person in case of failure and provide good results over the next few frames. Since we require a fast tracker, ASMS is a great choice. Besides speed we have found it to be quite robust, which allows the re-detection to be performed less frequently.

### 4.2.3 Optical Flow based Action Detection

After we are sure that we have a person in view and at known position in subsequent frames we employ a dense grid around the person. We use the output bounding box of the person detector as a reference for width and height. Optical flow is then computed between two adjacent frames within the dense grid. In a steady environment such as an immovable stationary camera, optical flow would only be generated by the persons gestures and we would simply need to define a threshold on a number of optical flow points and its magnitude that is generated for signaling an action in progress. However that is not the case with a moving drone platform. There can be sudden movements due to wind or errors in stabilization. Luckily such movements are not very common. There are still movements while the drone is trying to stabilize itself but they are smoother and usually the movements tend to return to a stabilized position. Each movement generates optical flow from one frame to another, so we need to subtract it from the optical flow that is generated by the gesture controlling person. In order to do this we use RANSAC [46] and keep only the outliers. Since the inliers generally represent background motion, or the motion of the drone, we are left with optical flow around the person.

**Optical Flow estimation**

In computer vision optical flow is normally used to estimate object motion from one image to the other with many applications. It is defined as an apparent motion between two images caused by an object or a camera and it is represented as a 2D vector field or in other words *optical flow* $\mathbf{u} = (u, v)^T$ is the visible displacement of a point in 2D. It begins at pixel location $p = (x, y)$ and ends at pixel location $p = (x + u, y + v)$. In most cases it is not identical to the actual movement. Therefore calculating optical flow aims at estimating the 2D motion. The optical flow equation is given by

$$0 = I_t + \nabla I \cdot [u\,v] \quad . \tag{4.9}$$

However if we look closely at Equation 4.9 we find that we have one equation and two unknowns $(u, v)$ and it is therefore not sufficient for estimating the apparent motion. This problem is very well known and it is called the Aperture problem. This problem refers to the fact that objects are viewed through a small aperture in effect limiting the view of a dynamic scene and therefore motion is ambiguous. The problem might be very well known to people waiting on a train about to depart. Since you can only view the outside world through a small window you might be mislead to believe that your train started moving but in reality it was the train on the next track that moved.

For action detection purposes we use **Lucas-Kanade** (L. K.) sparse optical flow. It is a simple technique for estimating the movement of interesting feature points in successive images. The goal of this method is to associate a movement vector $(u, v)$ to each of the feature points, obtained by comparing consecutive images. L. K. method makes three very important assumptions:

1. **Brightness constancy** meaning that we assume that the brightness from one image to the other of the tracked point does not change.
2. **Small motion** meaning that points are assumed not to have large displacements from one image to the other (they only exhibit small movements).
3. **Spatial coherence** meaning that points in a local neighborhood have similar movements.

L. K. solves the aperture problem with the last of the listed assumptions, namely spatial coherence. We assume that the flow is essentially constant in a local neighborhood around location $p$ with which we get more equations per pixel. We rewrite the optical flow equation as follows. Let $\mathcal{N}$ denote a $N \times N$ patch around a pixel $\mathbf{p}_i$. For each point $\mathbf{p}_i \in \mathcal{N}$, we can write:

$$0 = I_t(\mathbf{p_i}) + \nabla I(\mathbf{p_i}) \cdot [u\, v] \quad . \tag{4.10}$$

For example if we use a 5x5 window, we have 25 equations per pixel leading to more equations than unknowns. We can now write these equations in a matrix form $A\mathbf{u} = \mathbf{b}$, given by:

$$A = \begin{bmatrix} \nabla I(\mathbf{p}_1)^\top \\ \nabla I(\mathbf{p}_2)^\top \\ \vdots \\ \nabla I(\mathbf{p}_{N^2})^\top \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -I_t(\mathbf{p}_1) \\ -I_t(\mathbf{p}_2) \\ \vdots \\ -I_t(\mathbf{p}_{N^2}) \end{bmatrix} \quad . \tag{4.11}$$

An over-determined system can be solved as a least squares problem, $(A^\top A)\mathbf{u} = A^\top \mathbf{b}$. The solution is given by $\mathbf{u} = (A^\top A)^{-1} A^\top \mathbf{b}$ or $\mathbf{u} = A^+ \mathbf{b}$, where $A^+$ is the pseudo inverse of $A$. Therefore, combining information from a neighborhood of pixels, the L. K. method can resolve the inherent ambiguity of optical flow.

**Optical flow outlier detection**

Since optical flow is generated from interesting feature points in the background as well as the person gesturing to the drone, we need to use a method that will only care about those points generated by the person and ignore the ones generated by the movements of the background due to camera or drone movement. For this purpose we use a filtering process as proposed by [47].

The filtering is performed using a robust model fitting method. RANSAC, which is short for *random sample consensus*, is an iterative technique for estimating parameters of an assumed underlying mathematical model. RANSAC is given a set of data points (in our example interesting feature points), which are then split into inliers (points that follow the assumed model) and outliers (which do not follow the model and are considered noise).

RANSAC requires us to choose a number of iterations $N$. Usually the number of iterations is high enough to ensure that with probability $p_{in}$ at least one set of random samples does not include an outlier. Normally $p_{in}$ is chosen to be 0.99. We let $p$ represent the probability that the selected point is an inlier and $q = 1 - p$ a probability that the select point is an outlier. Number of iterations $N$ is then calculated according to

$$1 - p_{in} = (1 - p^{q_{min}})^N \quad , \tag{4.12}$$

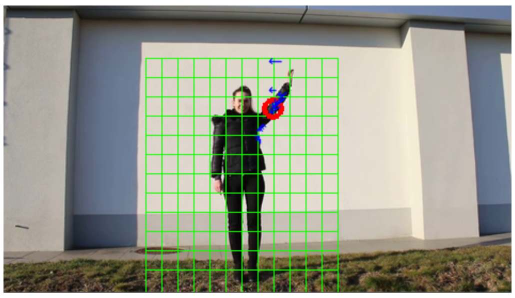
where $q_{min}$ is the minimum number of points required from which it follows

$$N = \frac{\log(1 - p_{in})}{\log(1 - (1 - q)^{q_{min}}} \quad . \tag{4.13}$$

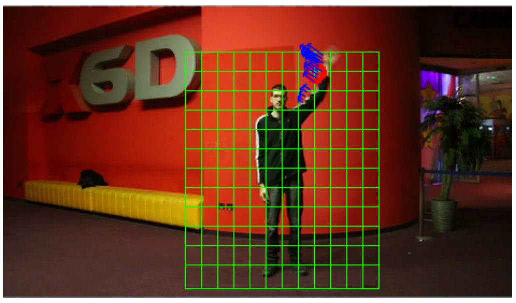
The RANSAC algorithm then works as follows

(i) Choose a model and determine $q_{min}$ points needed to describe the model.
(ii) Define a threshold for the inlier count (stopping criteria).
(iii) Randomly select $q_{min}$ points that are required to describe the model.
(iv) Solve for the parameters of the model.
(v) Apply the transformation to the set of all points.
(vi) Count how many points from the set of all points fit the model according to a predefined tolerance $\epsilon$.
(vii) If the number of inliers exceeds the predefined threshold we have found a good fit. Terminate.
(viii) Otherwise repeat steps $(iii)$ to $(vii)$ until we find the correct fit or reach $N$.

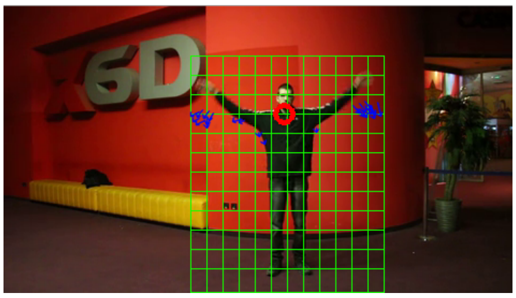
If we consider interesting feature points on the background to count as inliers and we choose the features on the person that is performing a gesture to be outliers, we are able to filter out the latter through the use of RANSAC. Therefore, we are able to subtract the optical flow generated by the movements of the drone from those generated by the person performing a gesture. The result of this is a clean optical flow accumulated around the person, within a predefined dense grid. Examples of optical flow outliers on some gesture videos are shown on figures 4.3a, 4.3b and Figure 4.3c.



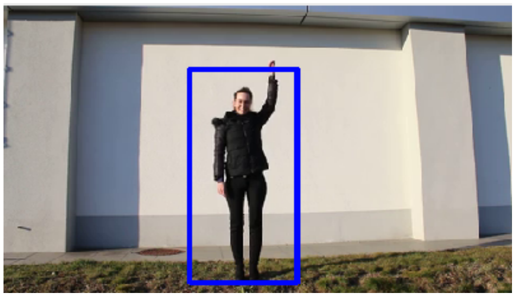
**(a)** Optical flow outliers example.



**(b)** Optical flow outliers example.



**(c)** Optical flow outliers example.



**(d)** ASMS tracker bounding box.

**Figure 4.3:** Examples of optical flow points (shown in blue) within the person's bounding box. Optical flow outliers provide the features for action detection. Note: Optical flow outliers are shown with a delay of 1 frame, so they might seem a bit shifted.

### Thresholding optical flow

After getting optical flow bound to the person and eliminating optical flow generated by the background, there are two steps left in the action detection pipeline. First we need to do something with the optical flow points to decide if there is action happening or not on a given pair of frames. For this part we experimented with various statistics, exploiting the dense grid around the person to determine the number of optical flow points in each segment and tried to define a metric to answer a yes when there is action happening and a no when there is no action. In the end we found the most simple approach to be the most effective. We simply count the number of optical points within the whole grid, disregarding the segments. And instead of finding the answer for each pair of frames, we extended it to more frames.

We measured each action duration as described in Section 6.2 in Table 6.2, and found that on average gestures in our dataset take 1.78 seconds. Since we decrease the fps to 15 instead of using 30, we have about 30 frames for one action. We decided to split the whole action into 6 separate **chunks**, *each spanning 5 frames*. This allows us to assemble a simple continuous descriptor, where each frame contributes a number of optical flow points generated by the person on the frame.

After assembling the descriptor of a chunk, it is sent to a linear SVM classifier, to determine if there was enough optical flow points to consider an action being performed throughout the 5 frames, surrounding the person. We decided to use a classifier and not a simple threshold due to the variety of descriptors that we have observed. During some actions some frames will have very little or zero optical flow points and then suddenly generate a lot. Some actions will have an approximately equal distribution of optical flow points throughout the action duration. And there can be some noise when no action is performed resulting in small numbers of optical flow being generated on each frame. This can occur if the person is completely stationary for some time and it is no longer an outlier but rather considered an inlier on the background.

Action, by our definition, now consists of *6 smaller chunks*, and we still need to decide if the whole action sequence was not a false positive, due to imperfect background from person filtering, or a false negative, when there is not enough outliers found. We also need to make this process continuous since we are constantly getting new chunks operating on a video stream. We implement this mechanism with *circular buffers*.

**Buffer implementation and management**

The circular buffer implementation ensures that we are able to process a video stream and check constantly if there is an action being performed. In case there are no actions performed, the video stream will continue. When there is an action detected, the system will be put into execution mode and the buffer containing frames on which there is a detected action will be forwarded for further processing.

There are two circular buffers with two different purposes. First buffer *circular image buffer* simply stores images (frames), which can then be forwarded. The second buffer **Circular classification buffer** contains the classification result of a chunk on which we performed optical flow thresholding. We push either a **1** in the classification buffer, when the classification algorithm predicts an action containing chunk or a **0** when it does not.

Circular image buffer stores chunks of frames up to some defined length at a time, each chunk consisting of a certain number of frames. Equally, the classification buffer stores classification results for the same number of chunks.

Let us consider two different examples and explain how they work in practice. We illustrate a classification buffer as an array of zeros and ones, where 1 represents a chunk, of some number of frames, classified as containing an action and 0 a chunk as not containing an action. First, let's take an ideal example. For the purpose of this example, let's set the buffer length to be 6. In an ideal example we would have 5 consecutive positive classifications (each of the 5 chunks was classified as containing an action), such that

$$\begin{bmatrix}\dots & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1}\dots\end{bmatrix} \quad ,$$

where the contents of the buffer are written in bold. Each time a new chunk is added to the buffer, the classification buffer will be checked if it reached the threshold for containing an action. For this example we decide that 4 out of 6 chunks must be classified as containing an action, to reach the threshold and signal a detected action. If that is the case, the system enters lock down and goes through further stages until an action is classified and executed.

Now we show a non-ideal example. If we find a classification buffer to be

$$\begin{bmatrix}\dots & 0 & 0 & 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0}\dots\end{bmatrix} \quad ,$$

our condition will not hold (there are 3 out of 6 chunks classified as containing an action) and so the threshold is not reached and we do not consider there to be an action contained on these frames. In that case the process goes on dismissing the most right chunk and a new chunk is pushed into the buffer at the most left end, checking again if our condition holds and so on.

**Conclusion**

We described the action detection pipeline that consists of rather simple but fast methods. We employ person detection to localize the person in a frame and afterwards track it with an extremely fast but still robust mean-shift tracker ASMS. The tracked person is re-detected every 30 frames to prevent tracker drift or failure due to sudden drone movements. After we are sure that we have the person localized we employ a dense grid around the person in which we calculate the optical flow using the Lucas-Kanade method. We filter the optical flow using RANSAC, so that only the person generated optical flow points remain. After filtering we count the optical flow points in 5 consecutive frames that form a chunk and classify it with a linear SVM, which decides if there was enough points, or not, to consider if the chunk as an action. Chunks are buffered into circular buffers, which provide continuous functionality and additional filtering of false positives and false negatives.

Due to the efficiency and speed of all of the above methods this part of the system has no issues running in real-time on CPU.

After we have detected an action the Circular image buffer is forwarded to the Pose Estimation module of the system, which is responsible for estimating the human pose, based on which, we can then compute important pose features.

## 4.3 Human pose estimation with Deep Learning

Human pose estimation has been and still is a very challenging task in computer vision. As we have described in Chapter 1, there has been a recent advancement in this field by harnessing the methods of deep learning. The state of the art methods are even able to run in real-time on powerful GPUs. Therefore this is the most computationally demanding part of the system. We describe two methods for two different purposes. First we describe Convolutional Pose Machines [13]. The advantage of CPM is its focus on single person pose estimation and therefore more precision. The second method that we use for our implementation is Real-time Multi-Person 2D Pose Estimation using Part Affinity Fields [14], which is a real-time method and is used for estimating the pose in real-time.

### 4.3.1 Convolutional Pose Machines

Convolutional Pose Machines (CPM) are one of many methods that recently exploited convolutional neural networks (CNN) for the task of articulated pose estimation. This movement started with using normal CNN structures, which didn't obtain good results and a shift to regression of image confidence maps began. Most researchers used graphical models in combination, which required hand-crafted energy functions and spatial probability priors in order to achieve better confidence maps. The advantage of CPMs is that they use no such things, but rather rely on large receptive fields to learn implicit spatial dependencies without the use of hand-crafted priors.

**CPM introduction**

The main idea behind CPMs is in using a sequence of CNNs that sequentially produce 2D belief maps. These produced maps are forwarded through multiple stages, where each stage refines the previous output. After producing belief maps, individual parts need to be linked together to form a representation of a person. Normally research would use graphical models or develop specialized procedures, which require a lot of post processing. Instead CPMs operate on belief maps in various stages and implicitly learn image-dependent spatial models of representation of the person. They found that using large receptive fields on belief maps and images is crucial for learning long range spatial relationships. Another contribution of CPMs was the reduction of a well known problem in deep CNNs (also called deep neural networks, which are simply networks with numerous layers) of *vanishing gradient*. Although this issue has been well addressed for classification tasks, they present a solution for structured prediction.

**From pose machines to Convolutional pose machines methodology**

Convolutional pose machines are an upgraded pose machines architecture introduced in [48]. The following is a description of the original methodology and how it became upgraded with deep neural networks. The goal of pose machines is to predict the locations of all body parts $Y = (Y_1, \dots, Y_P)$. In each stage predictors, $g_t$, are tasked at locating a body part with a belief, such that $Y_p = z$, where $\forall z \in Z$ given the set of features

$\mathbf{x}_z$ that were found in the image at location $z$ along with context information from the classifier in the previous stage t around the neighborhood of each $Y_p$. All relevant denotations for the following equations are presented in Table 4.2, for better readability. Belief values produced by a classifier in the first stage $t = 1$ can therefore be described with

$$g_1(\mathbf{x}_z) \to b_1^p(Y_p = z)_{p\in\{0\ldots P+1\}} \quad . \tag{4.14}$$

Classifiers output is therefore a score $b_1^p(Y_p = z)$ that assigned the part p to image location $z$. In subsequent stages beliefs for each part locations are predicted where features of the image data and contextual information from the preceding classifier around each $Y_p$ are computed according to

$$g_t(\hat{\mathbf{x}}_z, \phi_t(z, \mathbf{b}_{t-1})) \to b_t^p(Y_z = z)_{p\in\{0\ldots P+1\}} \quad . \tag{4.15}$$

Belief maps get refined stage after stage until a final result is given. In the original pose machines architecture image features were shared across stages and feature-engineered, which holds as well for context feature maps in order to capture spatial context across all stages. For predictors $g_t(\cdot)$ boosted random forests were used. Convolutional pose machines exploit the pose machines architecture but replace feature-engineered features and predictors with a deep convolutional network. Pose machines, and their corresponding CPM architecture, are shown on Figure 4.4. Multiple stages are used for enlarging the receptive field throughout the network, starting with a small patch around the pixel location. An intuitive way of understanding the network would be to view it as sliding a deep network over an image and for each patch, for $P + 11$ parts, regressing a belief map.

**Learning Spatial context and the importance of large receptive fields**

CPMs rely heavily on the learned spatial context features. One of the examples where this greatly benefits the pose estimation is a noisy estimate of the parts that are further away from the core of the body, such as elbows. By learning spatial relationships, an elbow is easier to localize, since it usually follows a shoulder. Therefore if the shoulder was predicted with

| | |
|---|---|
| $w$ and $h$ | Image width and image height. |
| $P$ | Number of all parts. |
| $\mathbf{x}$ | Image features computed / extracted from the image in the first stage. |
| $\hat{\mathbf{x}}$ | Image features computed / extracted from the image in subsequent stages. |
| $Z$ | Presents a set of all $(k, l)$ locations in an image. |
| $\mathbf{b}_t \in \mathbb{R}^{w \times h}$ | A set of beliefs of part $p$ for all image locations $Z$, so it follows $\mathbf{b}_t^p[k, l] = b_t^p(Y_p = z)$. |
| $\mathbf{b}_t \in \mathbb{R}^{w \times h \times P+1}$ | A set of beliefs of *all* parts. There are $P$ parts plus one part representing a background. |
| $Y_p$ | Refers to the $p - th$ location of a part, such that $Y_p \in Z \subset \mathbb{R}^2$. |
| $g_t(\cdot)$ | Multi-class predictor function, trained to predict the location of a specific part $p$ at each level. |
| $t$ | Denoting a stage, such that $t \in 1 \dots T$. |
| $\mathbf{x}_z \in \mathbb{R}^d$ | A vector of extracted features at a specific location $z$ in the image. |
| $\phi_{t>1}(\cdot)$ | Mapping from beliefs $\mathbf{b}_{t-1}$ to context features. |

**Table 4.2:** Denotation used in Equations(4.14) and (4.15) describing the CPM methodology.

high accuracy, the subsequent stage can use this context information to localize an elbow that might have been noisily detected in the previous stage. Authors of CPMs show that spatial context features greatly increase accuracy. This is possible by using large receptive fields. In CNNs larger receptive fields are achieved by either pooling, increasing the kernel size for convolutional filters or by increasing the number of convolutional layers. In CPMs the first stage belief maps are calculated using a rather small receptive field around parts, which drastically increases with each stage, as shown on Figure 4.4.

### Addressing vanishing gradients in deep networks

As the networks grow larger or "deeper" in terms of the numbers of layers, we are faced with *vanishing gradients*. While the number of intermediate layers is increasing, the magnitude of the gradients, which are back-propagated, is decreasing. Since this is a common problem that all CNN based pose estimation methods will need to address due to their deep network architectures, let's briefly overview how the CPMs deal with it. To solve this problem CPMs exploit the sequential nature of pose machines. They introduce a loss function calculated at the end of each stage $t$, as shown on Figure 4.4. The loss function minimizes the $L_2$ distance between currently predicted and ideal belief map for part $p$, given by $b_*^p(Y_p = z)$. Authors gave a good solution of introducing ideal belief maps by putting Gaussian peaks at ground truth locations for each part $p$. The loss function is given by

$$f_t = \sum_{p=1}^{P+1} \sum_{z \in Z} \left\| b_t^p(z) - b_*^p(z) \right\|_2^2 \tag{4.16}$$

and an overall loss function, which is the global objective of the network, is a sum of all losses at each stage, so

$$F = \sum_{t=1}^{T} f_t \quad . \tag{4.17}$$

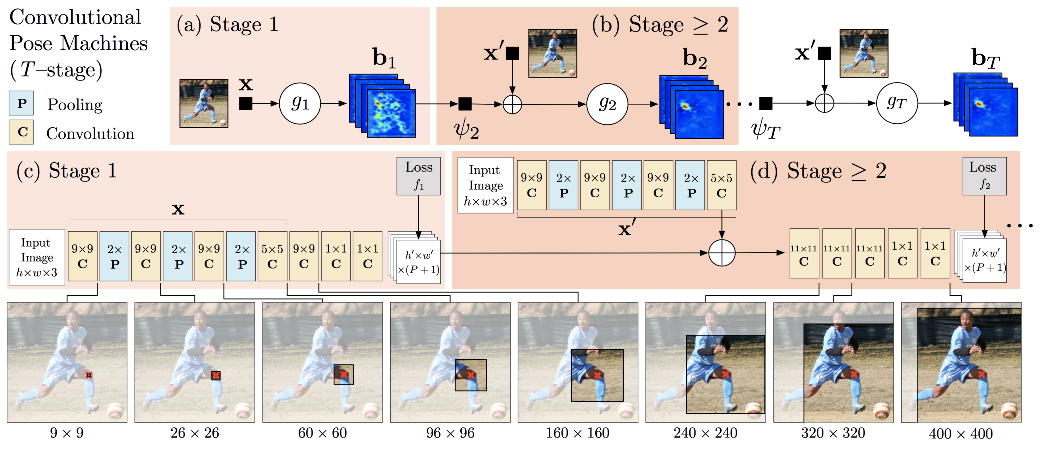


**Figure 4.4:** The architecture of CPM and a visualization of an increasing receptive field, as shown in [13].

An architecture of pose machines that operate only on image evidence for stage $T = 1$ is shown on (a) and (b), and the corresponding CPM architecture is shown on (c) and (d). Figures (b) and (d) show the architecture for subsequent stages $T \geqslant 2$, which incorporate belief maps from previous stages. Below (c) and (d) we also see how CPMs increase the effective receptive field, which allows the model to capture spatial-dependencies between body parts. In this example the receptive field is centered on the football player's knee. On (c) and (d) we can also see two additional loss functions, $f_1$ and $f_2$ at the end of each stage, which introduces local supervision to prevent the issue of vanishing gradient during training.

**Conclusion**

Due to their multi-stage refinement and spatial relationship learning, convolutional pose machines achieved extremely good results, significantly beating competitive methods on multiple datasets such as MPII Human Pose Dataset (with 10% improvement in accuracy, Leeds Sports Pose Dataset, and FLIC dataset (with a 14% improvement in accuracy). Therefore we chose this method for training and evaluating final classifiers, as will be described later. However, CPMs struggle with estimating the pose for multiple people in the same image and are not nearly fast enough for real-time performance.

### 4.3.2 Real-time Pose estimation

As opposed to a top-down approach such as CPMs, which have to be combined with a person detector for a full pipeline, we will now explore a bottom-up approach. Top-down approaches have generally had issues with ability to deliver fast computation and therefore it takes minutes to compute the estimated pose for one image. In addition, their performance strongly depends on the choice of a person detector, which causes the method to fail, if it has a false-positive or a miss-detection.

We chose a method called Real-time Multi-Person 2D Pose Estimation using Part Affinity Fields [14] (MPE-PAF) as the main method for real-time person estimation. It was the first method to deliver real-time performance for pose estimation and it works for multiple people at the same time. Like CPMs it is a method based on deep convolutional neural networks and inherits many ideas of CPMs, which were proven to have a very accurate pose estimation performance. MPE-PAF is a bottom-up approach that does not require a person detector. It has an architecture for jointly learning parts detection and parts association, which are then parsed with a greedy parsing algorithm to produce human pose estimation. This approach is not as accurate as CPMs but it does not perform much worse. Its main advantage is of course speed. Benchmarks were ran on a laptop version of NVIDIA GeForce 1080-GTX GPU, with which the authors obtain real-time performance of 8.8 FPS for a video with 19 people, making it an ideal choice for a real-time gesture control system.

**MPE-PAF Methodology**

The method takes as input a 2D color image and first predicts 2D confidence maps of body part locations and a set of 2D vector fields of part affinities, encoding the degree of association between parts using a convolutional neural network. The confidence maps are contained in a set **S** and part affinities in a set **L**. Then greedy inference is used for parsing the output of the CNN and finally outputting the 2D locations of joints. Similarly to CPMs, this method also simultaneously predicts confidence maps and provides spatial association of parts as shown on Figure 4.5, which authors refer to as Part Affinity Fields (PAFs, a set of 2D vector fields that encode locations and orientations of body parts). But the way of achieving this effect is slightly different. Instead of relying on large-receptive fields to learn the contextual information, they use two separate branches of the convolutional network. One branch is responsible for predicting the confidence maps and the other for affinity fields. Both branches adopt CPMs multi-stage architecture and in effect refine confidence maps and affinity fields throughout the stages. This requires a deep convolutional network, which means they are also prone to the problem of vanishing gradients. The method uses a fine-tuned VGG-19 [24] convolutional neural network, with which authors generate a set of feature maps **F**, which are used as input to the first stage $t_1$. The outputs of the first stage are $\mathbf{S}^1 = \rho^1(\mathbf{F})$, a set of belief maps and $\mathbf{L}^1 = \phi^1(\mathbf{F})$, a set of affinity fields. The subsequent stages feature maps **F** and outputs of the first stage are concatenated according to

$$\mathbf{S}^t = \rho^t(\mathbf{F}, \mathbf{S}^{t-1}, \mathbf{L}^{t-1}), \forall t \geqslant 2 \tag{4.18}$$

and

$$\mathbf{L}^t = \phi^t(\mathbf{F}, \mathbf{S}^{t-1}, \mathbf{L}^{t-1}), \forall t \geqslant 2 \quad , \tag{4.19}$$

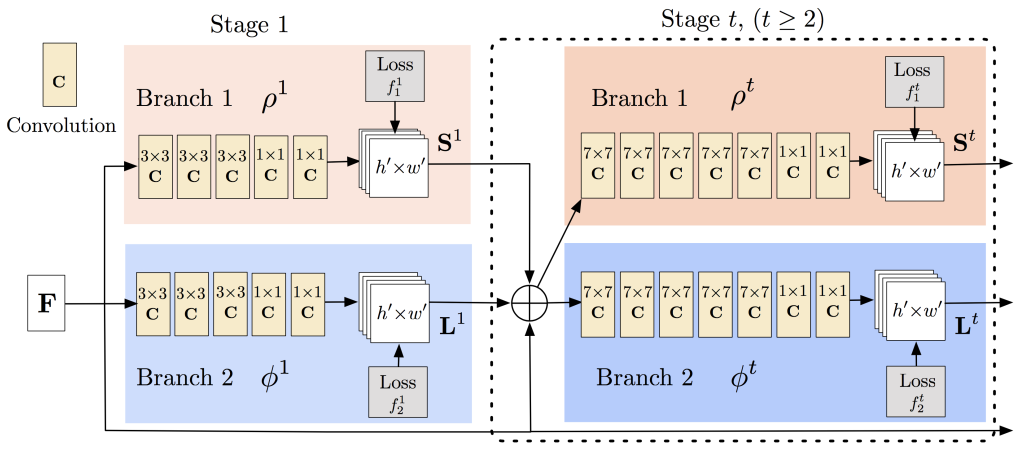


**Figure 4.5:** The architecture of MPE-PAF method as shown in [14].

As we have seen in CPMs, the architecture for MPE-PAF is also a two branch multi-stage CNN, where each stage in the upper branch predicts belief maps $\mathbf{S^t}$ and each stage in the lower branch predicts part affinity fields $\mathbf{L}^t$. Loss functions are used at the end of each stage to address the vanishing gradient problem and introduce intermediate supervision. Image features and predictions from both branches are merged together at the end of each stage to be used as input for the next.

to refine the predictions. To guide the learning process through the stages, two loss functions are applied at the end of each stage. Each branch has its own loss function. To compute the loss, authors use the $L_2$ distance between predictions and ground truth. The loss functions also use a weight, so that the learning can handle unlabeled people in the training data. The weight $\mathbf{W}$ is a binary mask and it is set to $\mathbf{W}(\mathbf{p}) = 0$, if the annotation is missing at location $p$, which prevents erroneously penalizing true positive predictions. With the use of loss functions at the end of each stage, similarly to CPMs, this method also deals with the problem of vanishing gradients. Loss functions that are used are the following:

$$f_{\mathbf{S}}^{t} = \sum_{j=1}^{J} \sum_{\mathbf{p}} \mathbf{W}(\mathbf{p}) \cdot \left\| \mathbf{S}_j^t(\mathbf{p}) - \mathbf{S}_j^*(\mathbf{p}) \right\|_2^2 \tag{4.20}$$

and

$$f_{\mathbf{L}}^{t} = \sum_{c=1}^{C} \sum_{\mathbf{p}} \mathbf{W}(\mathbf{p}) \cdot \left\| \mathbf{L}_c^t(\mathbf{p}) - \mathbf{L}_c^*(\mathbf{p}) \right\|_2^2 \quad . \tag{4.21}$$

As is the case with CPMs, this method also follows a global objective, which is a sum of both loss functions,

$$f = \sum_{t=1}^{T} (f_{\mathbf{S}}^t + f_{\mathbf{L}}^t) \quad . \tag{4.22}$$

To compute loss functions during training, ground truth has to be provided. Similarly to CPMs, authors use peaks positioned at 2D locations of annotated joints. The difference is in handling multiple-people on the same image and there has to be multiple peaks corresponding to each visible part $j$ for each person $k$. In this case ground truth confidence maps are computed according to

$$\mathbf{S}_{j,k}^*(\mathbf{p}) = \exp\left( \frac{\left\| \mathbf{p} - \mathbf{x}_{j,k} \right\|_2^2}{\sigma^2} \right) \quad , \tag{4.23}$$

where the spreads of the peaks are dictated by $\sigma$. These ground truth confidence maps are then aggregated with a maximum operator, so that

$$\mathbf{S}_j^* = \max_k \mathbf{S}_{j,k}^*(\mathbf{p}) \quad . \tag{4.24}$$

| | |
|---|---|
| $w$ and $h$ | Image width and image height. |
| $\phi^t$ | CNN network used for inference of belief maps at stage $t$. |
| $\rho^t$ | CNN network used for inference of part affinity fields at stage $t$. |
| $S$ | A set of 2D confidence maps for body part locations. The set contains $J$ confidence maps, one for each part, such that $\mathbf{S}_j \in \mathbb{R}^{w x h}$ and $j \in 1 \dots J$. |
| $L$ | A set of 2D vector fields of part affinities. The set contains $C$ vector fields, one per each limb, such that $\mathbf{L}_c \in \mathbb{R}^{w x h x 2}$ and $c \in 1 \dots C$. Each image location $\mathbf{L}_c$ encodes a 2D vector. |
| $\mathbf{S}^*_{j,k}$ | Ground truth confidence maps for each person $k$, where each body part has an index $j$ at location $\mathbf{p}$, such that $\mathbf{x}_{j,k} \in \mathbb{R}^2$. |

**Table 4.3:** Denotation used in Equations (4.18)-(4.24) describing the MPE-PAF methodology.

The maximum operation is used instead of the average so that peaks that are close to each other remain separated.

**Part affinity fields**

Authors of the method introduce *part affinity fields* or PAFs for associating parts with each other by encoding location and orientation information around the area of interest - the detected body joints. Each limb is represented as a 2D vector field, where each pixel in the area encodes the direction from one part of the limb to the other. With this method they are able to measure the association among pairs of body parts and also that they belong to the same person in the case, where there are multiple on a given frame. This makes the task of parsing the final human body pose easier.

**Parsing the final human pose using part affinity fields**

To assemble, or parse the final predicted pose, non-maximum suppression is first used to get rid of false positives for individual joints. Joints

estimates then need to be parsed into a correct configuration, which is a K-dimensional problem that is NP-hard. Due to the pair-wise association scores, which implicitly encode global context, thanks to the large receptive fields of PAFs, authors are able to introduce a greedy relaxation, which allows the parsing to be done very fast. Finding a single pair of parts for a limb, which presents a sub problem, the problem reduces to maximum weight bipartite graph matching [49], where body part detection candidates are nodes and a set of all possible connections between pairs of detection candidates are edges. Solving this problem produces a subset of edges chosen in a way that no two edges share a node or in other words, no two same type limbs (a limb type is for example left ankle, right elbow, etc.) share a part. To solve this problem authors use the Hungarian algorithm [50].
Then full-body poses of multiple people need to be found, for which the authors use a greedy relaxation by taking the minimal number of edges that form a spanning tree skeleton of a human (which forms a top down structure, in effect reducing the number of edges) instead of a fully connected graph. They also further divide the matching problem into smaller sub problems. In this way minimal greedy inference is used to find very good approximations of the global solution with very low computational costs. A more interested reader can refer to the original paper [14], to find more details about the method.
The result of this method are limb connection candidates for each type of a limb, which allow for the final human pose to be assembled, even if there are multiple people on the same image, in real-time.

**Conclusion**

The method described above in Section 4.3.2, provides real-time computation of a human pose given an image by exploiting the structure of Convolutional Pose Machines, presented in Section 4.3.1 while combining it with an extremely fast pose parsing using PAFs. Speed does come with a sacrifice on accuracy, but it is not decreased too much, and the method is still achieving SOTA results.

## 4.4 Gesture classification

The result of action detection is a set of frames that contain an action sequence. This set of frames is then forwarded to the human pose estimation module of the system. After the pose is estimated on those frames, we are given a set of body joints for each frame. Obtaining the $x$ and $y$ positions for each body joint allows for certain features to be computed. The descriptor, which is a combined set of various features, computed on the body joint positions, is referred to as *Pose Features*. This descriptor combines positional features and relational features that describe geometry between combinations of joints. It was shown that a combination of these two types of features produces the best results for gesture classification [51].

### 4.4.1 Calculation of Pose Features and assembly of the frame descriptor

All joint positions are normalized to $[0, 1]$. Coordinate $x$ is normalized to $[0, 1]$ in regard to $W$, width of video frame, and coordinate $y$ is normalized in regard to $H$, height of the frame. After normalization of joint positions, pose features are computed. We use two datasets for the development and evaluation of the algorithm and the number of joints differs between the two. One is annotated with 15 body joints and the other with 14. For this example we will assume that we are computing features on the former, which is annotated with 15 joints per frame.

We calculate the following 5 types of features from the estimated pose:

1. position features,
2. distance features,
3. orientation features,

4. angle features and
5. temporal features.

*Position features* are simply the normalized joint coordinates. Since there are 15 joints, we end up with 30 positions (15 for $x$ and 15 for $y$).

*Distance features* are computed between pairs of joints for each frame of an action sequence. Since we have 15 joints, we first have to find all combinations of joint pairs. We denote joints with $j_i$ and its coordinates with $x$ and $y$. There are $C_2^{15} = 105$ such combinations. For joint pairs the distance between each joint is then computed using $dx = (j1_x - j2_x)^2$ and $dy = (j1_y - j2_y)^2$ in $d = \sqrt{dx + dy}$.

Next we compute *orientation features* between joint pairs, on each frame of the action sequence. Again, we have $C_2^{15} = 105$ combinations. We compute the orientation of the vector connecting two joints. First we calculate $dx = j2_x - j1_x$ and $dy = j2_y - j1_y$. Then we calculate the orientation with $\alpha = \arctan2(dy, dx) \cdot (180/\pi)$. We then adjust the orientation according to the neck to belly joint orientation, therefore $|\alpha| > 180^\circ$ with $360^\circ - |\alpha|$.

Then *angle features* are computed. Angle features are represented by the inner angle of two vectors that span triplets of joints. Since we are using triplets of joints, we have $3 \cdot C_3^{15} = 1365$ combinations. For each triplet we let vector between two joints $j_1$ and $j_2$ be $\mathbf{a} = [(j1_x, j1_y), ((j2_x, j2_y)]$ and the vector between joints $j_2$, $j_3$ be $\mathbf{b} = [(j2_x, j2_y), ((j3_x, j3_y)]$, then calculate the angle according to

$$\phi = \arccos\left(\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| \cdot |\mathbf{b}|}\right) \quad . \tag{4.25}$$

Finally *temporal features* are computed. Temporal features consist of differences of distance features, differences of orientation features and differences of angle features between two adjacent frames. We can select over how many frame pairs we wish to compute the temporal features.

In total this gives us $(C_2^{15} + C_2^{15} + 3 \cdot C_3^{15})$ combinations or $(105 + 105 + 1365) = 1575$ relational features per frame, which are combined with 30 positional features and additional 1575 differences of relational features, calculated with temporal features, giving us a total of 3180 features per frame, which make up the Pose Features frame descriptor.

### 4.4.2 Bag of Words learning

Bag of Words (*BoW*) is a very popular method for image or visual data classification, which uses discriminatory features derived from images or parts of images as words. These features can usually be sharp edges, corners, various color combinations, and so on, typically extracted as patches on which descriptors such as SIFT and SURF are calculated. The bag is a term for the model, which consists of important words called a codebook, vocabulary or dictionary. They are usually represented by histograms, where each bin corresponds to a word. These codebooks are usually created by applying K-means clustering to an accumulated database of extracted features from a large number of images, that are similar to the images we wish to classify. K-means groups similar features together and finds clusters, represented by centroids (cluster centers), which then represent the codewords in the codebook. When we receive a new example (in our case an image), we extract the patches and the patch features using the same method and calculate the distance to all the codewords that we have collected in the codebook. We take the closest one for each patch, effectively "assigning" it to a cluster, and increment the corresponding bin in the histogram. We end up representing an image with a histogram, that counts the number of each important word that appears on the image, called a *visual word*.

The drawback of this method is that there are no spatial relationships modeled into the representation or in other words, we do not know where on the image features came from or how they relate to each other.

After assembling the BoW codebook, the method's speed depends on the type of descriptors used, which is perfect for Pose Features, since they are a very fast descriptor to compute.

### 4.4.3 Bag of Words learning of Pose Features

Our implementation of BoW learning is very similar to the method described above. The method is split into two stages, where first stage is used for training, or in other words assembling the codebook, and training the SVM. The second stage provides on-line inference.

For training we split each video that contains an action into frames and consider each frame in place of a patch described in the general method. We then proceed to compute Pose Features on each frame, which returns a descriptor. The number of descriptors depends on the length of the input video (number of frames in the video). After computing Pose Features we perform K-means clustering to quantize the descriptors into clusters. After the centers of clusters are returned we calculate the distance of each frame descriptor to each cluster and find the closest cluster center. Each cluster is represented a as bin in the histogram and upon computing the closest cluster we increase the corresponding bin value by 1. After this is done for each frame, we end up with a histogram representing the whole video (action). The histogram is then normalized. Histograms are collected into a database and used as features for training the SVM. We use a radial basis function (or *RBF*, described in Equation (4.26)) for the kernel

$$K(x_i, x_j) = exp\left(\frac{||x_i - x_j||^2}{2\sigma^2}\right) \quad , \tag{4.26}$$

where $x_i$ and $x_j$ are two feature vectors.

The procedure is extremely similar for the inference stage. After the pose is estimated on frames that were deemed to contain an action, Pose Features are computed on each of them and the closest cluster center is found for the assembled descriptor. Using the same procedure as above, a histogram is assembled, normalized and then sent to the SVM for classification, which produces the result - a given action by the user.

### 4.4.4 Conclusion

Using the Pose Features descriptor with a BoW approach makes up the final part of the system, which produces the final result - a classified gesture that is sent to the drone as a command.

# Chapter 5

# Gesture control system implementation

## 5.1 Robot Operating System

Since the methodology is split between separate modules, we wanted to keep the implementation modular as well. We have also mentioned that the GPU on the on-board dedicated computer on the drone is not supported by deep learning frameworks. Therefore a system that can be easily distributed among computation units is what we need. In this way it does not matter if the whole pipeline is running on the same computer or is split between two or more. It also allows us to be very independent of the hardware layout. For example we can currently run the system distributed between a drone and a PC, where the drone can execute all parts of the system apart from the human pose estimation, that requires a special GPU. However, if we find that a mobile computer, which has such a GPU, exists, we can easily switch to a pipeline being run completely on the drone itself.

What follows is a short overview of the *Robot Operating System*, which is capable of what we have described above, and therefore we have decided to use it for implementing the gesture control system.

### 5.1.1 Introduction to ROS

The Robot Operating System or *ROS* is an open-source framework, that acts as a meta-operating system, combining a publish-subscribe messaging infrastructure for distributed systems along with a multitude of tools, usually found in operating systems, such as configuring, logging, testing, visualizing and running distributed systems. ROS is well suited for robots as it provides a number of packages for various sensors such as cameras, sonars, and so on, as well as packages for coordinating sensors with the body of the robot through coordinate system transformations and their interaction. In a broader sense ROS provides tools and libraries in the form of packages and stacks that can easily be integrated into the framework, to help developers develop applications for robots.

### 5.1.2 Nodes in ROS

ROS uses a *Computation graph* that is a peer-to-peer network of processes, with basic units called *nodes*. The root of the Computation Graph is the ROS *Master*, which enables the nodes to see each other and provides the infrastructure for messaging.
*Nodes* are processes, that execute programs and are the key to a distributed system. Nodes can be programs that control sensors or motors, provide high-level functionality such as action detection and so on. A robot system usually comprises of many such nodes. ROS currently supports nodes written in either C++ or Python.

### 5.1.3 Messages in ROS

*Messages* are a simple way of communication between nodes in ROS. Developers can define their own messages by specifying data types or data structures. Nodes pass messages to each other via *topics*, which are a publish / subscribe system. Nodes send out messages, which are *published* on specified topics. Names of the topics are also used to identify the contents of the messages. If a node wishes to receive a certain type of data it can *subscribe* to that topic and the data will be forwarded when it is published on it. Nodes can publish on multiple topics and can also subscribe to multiple topics.

In addition to the publish / subscribe system, which is a many-to-many way of data transport, ROS also supports *Services*. With services nodes can use a one-to-one data transport using a message pair of a request and a reply. A node can send out a request and wait for a reply that is sent back by another node. ROS provides this functionally to the programmer in a similar way as a remote procedure call.

### 5.1.4 Conclusion

ROS offers a modular processing functionality while also providing a messaging system for passing data between the distributed nodes over a network and is therefore a good fit for the implementation of the gesture control system running on a mobile platform, that is in a sense, a robot system. It also allows us to use a dedicated PC with a specific GPU that is required for the computation of human pose estimation in real-time in combination with the drone. Last but not least it makes our implementation independent of the current hardware architecture, as the nodes can either be distributed between two or more computation units or be executed on a single unit - the drone.

## 5.2 Implementation

In this section we will describe the system implementation in detail. We will describe the system architecture as implemented in ROS and then describe each node separately.
We implemented the system using the programming language C++ and the meta-operating system ROS, relying heavily on frameworks *OpenCV*,

an open source computer vision library, and *Caffe*[52], a deep learning framework.

### 5.2.1 System architecture overview

The first part of our implementation is the *Video Node*, which is responsible for providing a video stream from the drone's RGB camera, while in operation. This node provides data to the *Action Detection Node*, which analyzes the incoming video stream for actions and segments it accordingly. Frames on which the action was detected are forwarded to the *Pose Estimation Node* running on a dedicated PC, with an NVIDIA GPU. Its task is to estimate the human pose estimation using deep learning algorithms. After pose has been computed for each frame, the estimated joint positions are sent to *Gesture Classification Node*, which computes Pose Features and classifies them as gestures. Detected and classified gestures are sent to the *Command Center Node* that translates the user's gesture into a command and forwards it to the drone's autopilot for execution. After a command has been acknowledged and successfully executed, the whole system runs in a continuous mode, waiting for more gestures.

For better visual representation a diagram, showing all ROS nodes and how they are connected, is shown on Figure 5.1.

### 5.2.2 Video Node

This node is responsible for opening, receiving, handling, configuring and forwarding a video stream from the drone's RGB camera, frame by frame. This node is subscribed to *image_raw* topic, where the drone's camera stream is published in a raw format. We can set the desired frame rate and control basic camera properties. Besides listening to the *image_raw* topic, the node is also listening to *stream_enable* topic, used for controlling the stream forwarding to the next node, which is stopped when the gesture control system is in the process of executing a command and continued afterwards.

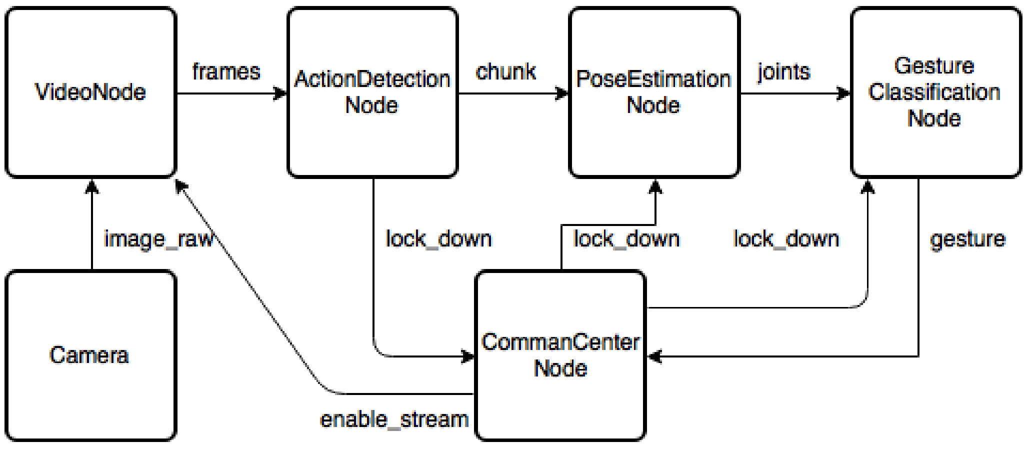


**Figure 5.1:** The diagram shows all ROS nodes and how they are connected via topics. Rectangles represent individual nodes and arrows represent topics, via which the messages are exchanged.

We have configured the camera to provide 15 FPS instead of 30 FPS, to reduce the load on the system and allow for faster processing.

**Subscribed topics** (Input):

1. *drone/camera/image_raw* - receiving raw frames from the camera
2. *drone/video/stream_enable* - stream control, on or off

**Publishing on topics** (Output):

1. *drone/video/frames* - published sequence of pre-processed frames from the drone RGB camera

### 5.2.3 Action Detection Node

Upon receiving frames from the *Video Node*, *Action Detection Node* will first detect a person using a person detector to get the estimated location and the bounding box of the person. Then it will start tracking the person using the *ASMS* tracker. Re-detection of the person is done every 30 frames, which we found to be sufficient due to *ASMS*'s robustness. Individual frames will be pushed back into the circular buffer to form chunks. We are using a length of 5 frames to represent one chunk. Optical flow is then calculated between pairs of frames according to the method described in 4.2, to collect the number of optical flow points generated by the person. After a chunk has been assembled, a descriptor consisting of numbers of OF points is calculated and sent to the trained SVM, which then classifies the chunk to see if it contains an action or not. Since we have found the actions to take approximately 2 seconds, each action takes 30 frames (we receive 15 frames per one second) or 6 chunks. If there is no action present on chunks, the work flow continues undisturbed. If the assembled Classification buffer (as described in Section 4.2) then consists of enough chunks with action detected this node will first publish a "lock down" signal, which will freeze all nodes up to the *Pose Estimation Node*, to free up computational resources as well as to prevent overflowing the system. The "lock down" signal is published on a special topic, so that all running nodes in the system are aware of it. Then the node will publish the Image buffer containing 6 chunks (30 frames) with an action, as a sequence of frames.

**Subscribed topics** (Input):

1. */drone/video/frames* - receiving pre-processed frames from *Video Node*

**Publishing on topics** (Output):

1. */drone/ros_action_segmentation/chunk* - publishes frames that contain an action
2. */drone/ros_action_segmentation/lockdown* - a lock down signal to freeze certain nodes

### 5.2.4 Pose Estimation Node

After detecting an action with the *Action Detection Node*, the system is in a "lock down" up to the *Pose Estimation Node*, which is responsible for estimating the human pose on a sequence of frames. This node receives 6 chunks, each of length 5 frames, and for each frame estimates the joint positions using a deep learning method as described in Section 4.3. The deep learning method is implemented using *Caffe* [52], a deep learning framework by Berkeley AI Research with a C++ backend. Unfortunately this framework (or any other) at this time does not support ARM based GPUs, such as the one on the dedicated on-board computer on the drone, we are forced to use a separate PC that is connected to the same network as the drone. This is very well handled with ROS, as nodes are used exactly for the purpose of distributed system. Therefore *Pose Estimation Node* runs on a PC called a "ground station", that has an NVIDIA GPU. To estimate the pose we use a model trained on the MPII dataset [53], which outputs 15 body joints. After all frames have estimated joint position, they are concatenated and published on */groundstation/pose_estimation/joints*.

**Subscribed topics** (Input):

1. */drone/ros_action_segmentation/chunk* - receiving a chunk of frames that contain an action for further processing
2. */drone/ros_command_center/lockdown* - a lock down signal to freeze certain nodes

**Publishing on topics** (Output):

1. */groundstation/pose_estimation/joints* - estimated human body joints locations on each of the frames

### 5.2.5 Gesture Classification Node

The *Gesture Classification Node* receives the estimated join position from the *Pose Estimation Node* and computes the *Pose Features* as described in Section 4.4. After calculating the BoW histograms, they are classified with a pre-trained SVM. For the purpose of training the SVM we created our own dataset of gestures, which is described in Section 6.2. A trained SVM predicts the final gesture, which is sent to the drone as a command. The node will pause and wait until the whole process of detecting an action and estimating the pose happens again.

**Subscribed topics** (Input):

1. */groundstation/pose_estimation/joints* - a vector of estimated human body joints locations
2. */drone/ros_command_center/lockdown* - a lock down signal to freeze certain nodes

**Publishing on topics** (Output):

1. */drone/gesture_classification/gesture* - the final predicted gesture, which the drone will interpret as a command

### 5.2.6 Command Center Node

The *Command Center Node* is sort of a master node, connecting all previously described nodes and the node, which also communicates with the drone's autopilot.
It is responsible for triggering the streaming of the camera feed on the drone and commanding the video pre-processing node *Video Node*. It is also responsible for translating the output of the *Gesture Classification Node* and providing it to the drone as an actual command, that the autopilot understands. Since the drone is equipped with a GPS, gestures are translated into GPS points relative to the drone's current GPS position. The *Command Center Node* also listens for a signal from the drone, that the command has been executed and the drone has returned to the original position.

**Subscribed topics** (Input):

1. */drone/gesture_classification/gesture* - the final predicted gesture, which the drone will interpret as a command

**Publishing on topics** (Output):

1. */drone/ros_command_center/lockdown* - a lock down signal to freeze certain nodes

### 5.2.7 Conclusion

In this section we described how the system is integrated and implemented in ROS with separate nodes for the system modules. We described the inputs and outputs to each of the separate nodes, starting with Video Node, Action Detection Node, Pose Estimation Node, Gesture Classification Node and ending with a Command Center Node. ROS provides the required infrastructure for the nodes to be deployed as a distributed or a centralized system and provides the necessary communication between them.

# Chapter 6

# Gesture control system evaluation

## 6.1 Introduction

In this chapter we evaluate different components of the gesture controlling system. For this purpose we first describe two datasets, JHMDB and DS2017 that aided in the development of the system. We then evaluate action detection on the DS2017 dataset, we review the evaluation of the real-time pose estimation and analyze its advantages and short-comings. We also evaluate the pose estimation on the DS2017 dataset and provide examples. Finally we evaluate the Pose Features with Bag of Words gesture classification method on both JHMDB and DS2017.

## 6.2 Datasets

In this section we describe in detail the datasets, that we used for evaluations and development. The described datasets offer a solid basis, on which we can test action detection, pose estimation and gesture recognition. One of these two datasets was recorded, annotated and prepared by us, for the purpose of developing and evaluating the system for gesture control.

### 6.2.1 JHMDB Dataset

Joint-annotated Human Motion Data Base or *JHMDB* [51] is a dataset prepared for the purpose of action recognition providing both low-level features (dense optical flow) and high-level features (estimated human pose with annotated joints). It is comprised of 928 short videos containing 21 distinct actions or in other words, it provides on average 44 videos per action. The actions include *shoot bow*, *clap*, *chew*, *walk*, *sword exercise* and others. They were collected "in-the-wild", which usually means online videos, without any controlled environments, offering a wide range of backgrounds, illuminations, cameras and so on. Actions are always performed by a single person, which is usually the only person in the video (some videos are an exception). Authors annotated each frame by trying to fit a 2D puppet model to the person by hand using an annotation tool created for this purpose. These annotations provide scale, segmentation, joint positions, a puppet mask and a puppet flow. Authors removed the before and after action frames, where the action is not being performed, to focus on the actions. However, this is bad news for us, since we can not evaluate action detection using this dataset, due to having only short clips with actions. There are a total of 31,838 annotated frames, providing a solid evaluation basis for action recognition. An interested reader can refer to the original paper [51] to learn more about the annotation process.
An example of an action present in the dataset is shown on Figure 6.1. We can see a person *shooting a bow*. Notice that the person is not fully visible. The visible part is only the persons upper body. Another example is shown on Figure 6.2, where a bartender is mixing a drink. As is the case before, we can see that the person is not fully visible. This is an example for action *pour*.

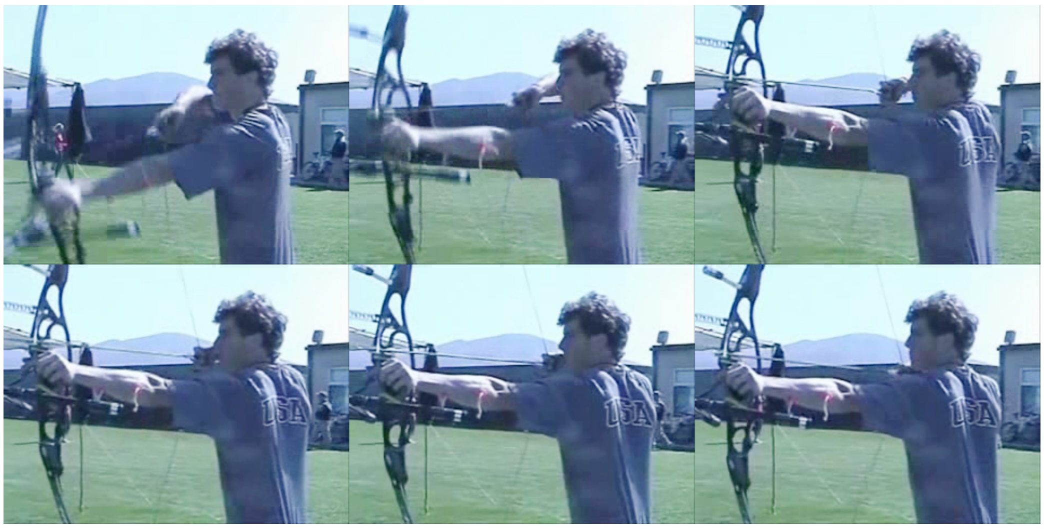


**Figure 6.1:** An example of action **shoot bow** from the dataset JHMDB featuring an archer, shooting an arrow. The image sequence progresses from the upper left corner to the bottom right corner.

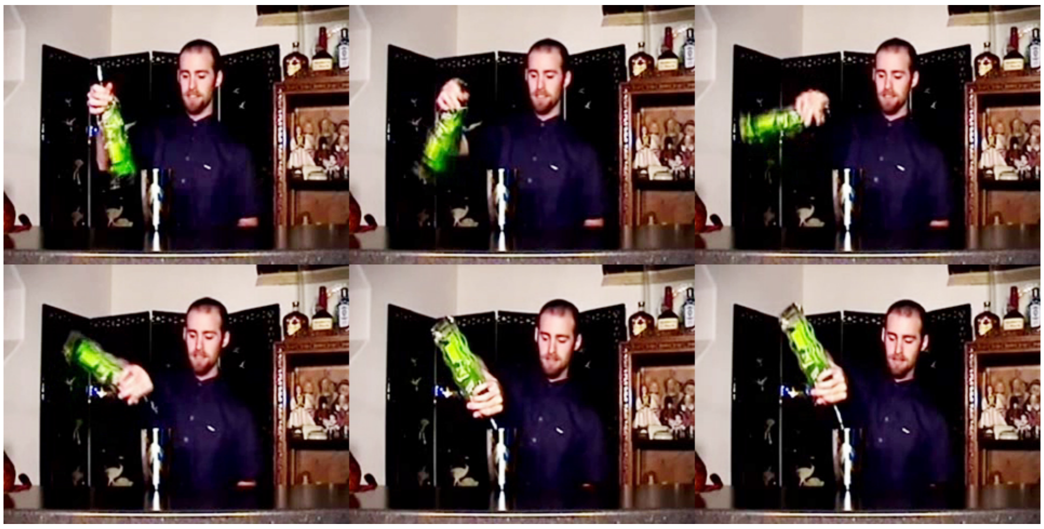


**Figure 6.2:** An example of action **pour** from the dataset JHMDB featuring a bartender, preparing a drink. The image sequence progresses from the upper left corner to the bottom right corner.

### 6.2.2 DS2017 Dataset

For the specific purpose of developing and evaluating the gesture control system, we have created our own dataset that we have named *DS2017*. We have decided to use 4 main gestures for controlling the drone, namely *up*, *down*, *left* and *right*. For these 4 gestures we have created a *controlled gestures* dataset and, having the same 4 actions (up, down, left and right) an *intuitive* dataset, which form the two main categories of DS2017. Each of these two categories is further split in a set of videos with a *steady camera* and the second with a *moving camera*, simulating the movements of the drone. Therefore we have 4 sub-sets in total,

1. Controlled-steady set,
2. controlled-unsteady set,
3. intuitive-steady set and
4. intuitive-unsteady set.

**Composition of the DS2017 dataset**

The DS2017 consists of videos of 20 people, where each person performed 4 gestures, twice in a row, in each video, producing in total 8 gestures per video per sub-set, resulting in a total of 32 gestures per person. This results in 4 longer videos per each person. Therefore the whole DS2017 dataset is comprised of 640 gestures performed in 80 videos.
To get a better overview Table 6.1 shows a summary, where we specify the number of videos recorded for each category and gesture and their combinations.

**Annotation**

Each video was first split into clips of gestures, similar to the JHMDB action clips. This provides the labeling for the gestures (up, down, left, right). With the help of these clips we also labeled each 5 consecutive frames, if they contain a gesture or not, resulting in a ”continuous” video being labeled, which serves as an evaluation for the action detection part

| DS2017 Summary | | | | | |
|---|---|---|---|---|---|
| **Con/Int** | Left | Right | Up | Down | **Total** |
| Steady | 40 | 40 | 40 | 40 | 160 |
| Unsteady | 40 | 40 | 40 | 40 | 160 |
| | 80 | 80 | 80 | 80 | **320** |

**Table 6.1:** A summary of total videos recorded per gesture (left, right, up, down) per category (steady, unsteady) for both the control and intuitive gesture set.

of our system.
Then all short clips were organized by categories, as described in the List 6.2.2. Besides these categories there is also an *intuitive* and a *controlled* set, which include steady and unsteady videos combined.
In summary every video is annotated for action detection with 5 frames granularity and split into gestures then categorized in one of the 4 main categories, thus providing a basis for the evaluation of the gesture control system.

**General assumptions of the dataset scene**

In order to not over complicate the dataset and to provide a good evaluation basis, the videos are more controlled in terms of background and illumination, as well as taken in the same image format with the same camera. Unlike JHMDB, which features videos captured "in the wild", we limited DS2017 scenes to relatively homogeneous backgrounds, with equal illumination throughout the video. Although there are some examples where this does not hold. There is always a single person in the video, which is performing a gesture and in general the person does not move around on the scene. Scenes changed mostly from person to person however, giving us a wide range of backgrounds.

**Controlled gestures**

The purpose of a controlled gesture set is to have a very distinct set of gestures, that are very exuberantly performed in a standardized manner, potentially resulting in more optical flow points and have similar Pose Features descriptors, even when different people perform the gesture. It is meant to serve as a standardized controlled experiment, on which the algorithms would be evaluated. The gestures were partially inspired by NATOPS aircraft signaling gestures, similar to how authors of [22] built their dataset. The particular gestures we chose, up, down, left and right, are simple and useful gestures for controlling a drone. This set of gestures is extremely different to the actions performed in the JHMDB dataset (for example ″shoot bow″) and is more relevant for our application.
Controlled gestures are shown on Figure 6.3, an example of gesture ″Up″ and on Figure 6.4, an example of gesture ″Left″. Gesture ″Right″ is shown on Figure 6.5 and finally gesture ″Down″ is shown on Figure 6.6.

**Intuitive gestures**

Unlike the controlled gestures we wanted the intuitive gestures to be more natural to how a human would gesture a drone, to perform a specific action. For that purpose we re-created the four gestures (up, down, left and right) and imagined them to be intuitive and more fluent. When capturing the dataset these actions were not thoroughly regulated or explained to the people who would perform them in order to encourage variability of gestures from person to person. These sub-sets represent a more challenging part of the dataset.
Intuitive gestures are shown on Figure 6.7, an example of gesture ″Up″ and on Figure 6.8, an example of gesture ″Left″. Gesture ″Right″ is shown on Figure 6.9 and finally gesture ″Down″ is shown on Figure 6.10.

**Steady dataset**

Steady sub-sets were taken on a stabilized camera, so that there is no movement of the background and the only thing that is moving is the

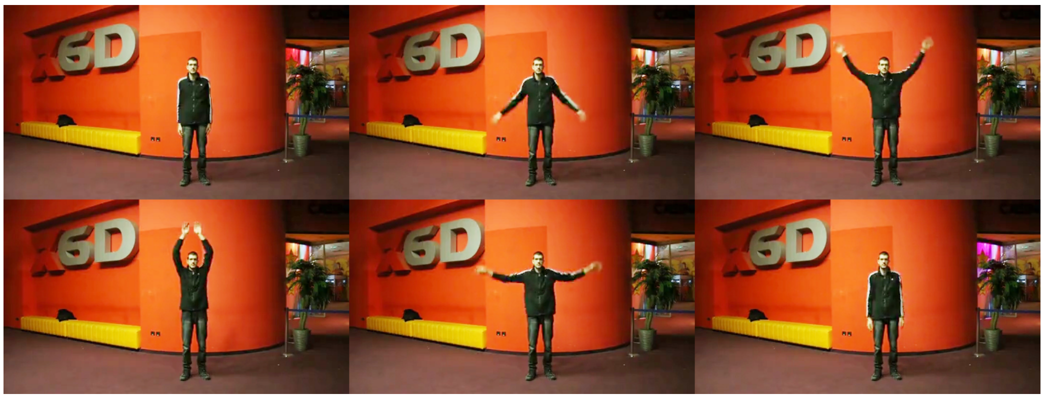


**Figure 6.3:** An example of **controlled** gesture **up** from the dataset DS2017. This example features one of the more complex backgrounds as well as various illumination on the scene. However, the person is still relatively easily distinguishable from the background. Notice that the person is visible in full at all time of the action. This is an example of a stabilized camera. The gesture starts in a neutral position and continues into both arms being lifted in a half circular motion, connecting above the person's head, then lowering back down in the same half-circular motion to the starting position. The image sequence progresses from the upper left corner to the bottom right corner.

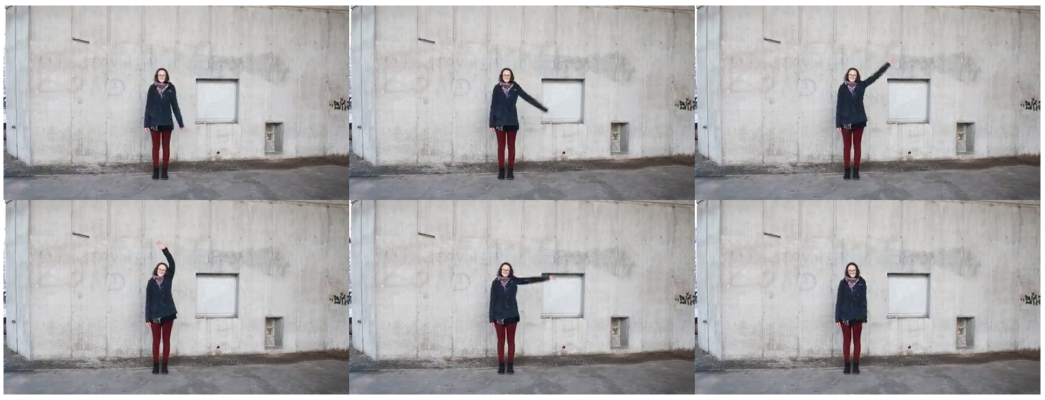


**Figure 6.4:** An example of **controlled** gesture **left** from the dataset DS2017. This example features one of the more simple backgrounds with a clearly distinguishable person. Again the person is visible in full throughout the entire gesture. This is another example of a stabilized camera. The gesture starts in a neutral position and continues into the left arm being lifted in a half circular motion, reaching a near vertical position above the person's head, then lowering back down in the same half-circular motion to the starting position.

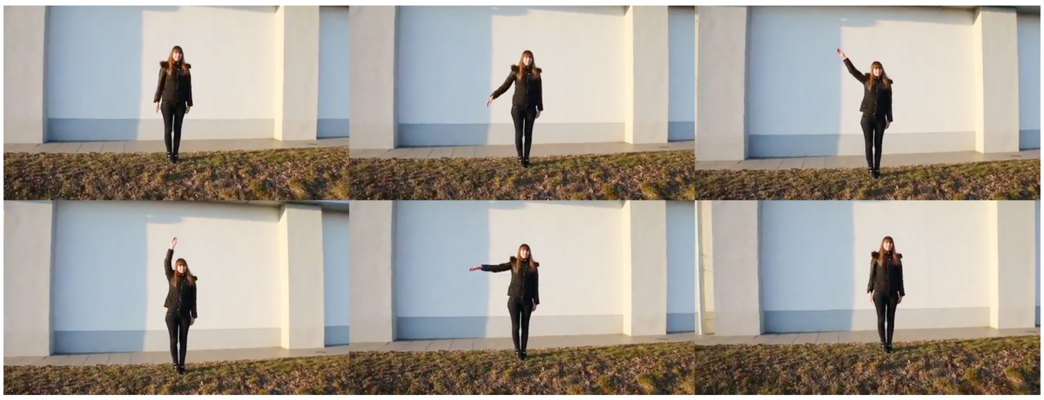


**Figure 6.5:** An example of **controlled** gesture **right** from the dataset DS2017. This example features a hard shadow on the background with a clearly distinguishable person. Again the person is visible in full throughout the entire gesture. This is an example of an unsteady camera. The gesture starts in a neutral position and continues into the right arm being lifted in a half circular motion, reaching a near vertical position above the person's head, then lowering back down in the same half-circular motion to the starting position, similar to the gesture "left".

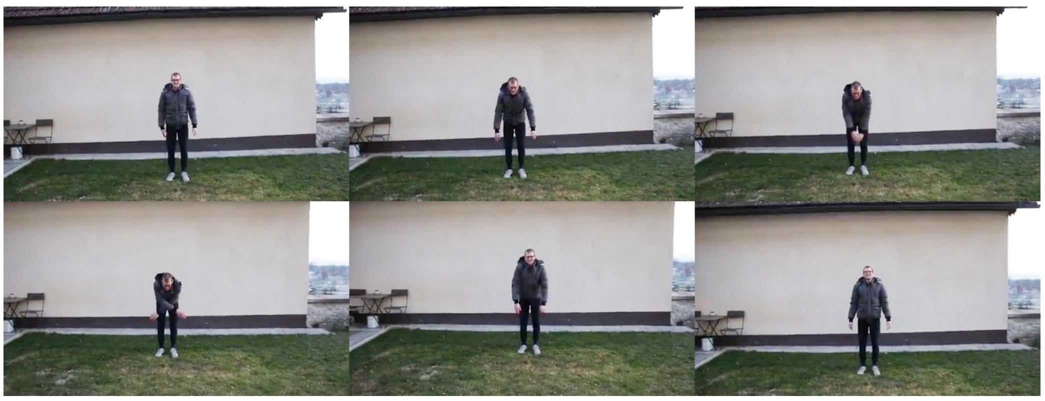


**Figure 6.6:** An example of **controlled** gesture **down** from the dataset DS2017. This example features a very simple background. Again the person is visible in full throughout the entire gesture. This is another example of an unsteady camera. The gesture starts in a neutral position and continues into the upper body being lowered and the arms criss-crossed in front of the person's legs. Then both the upper body and both arms returned into a neutral position.

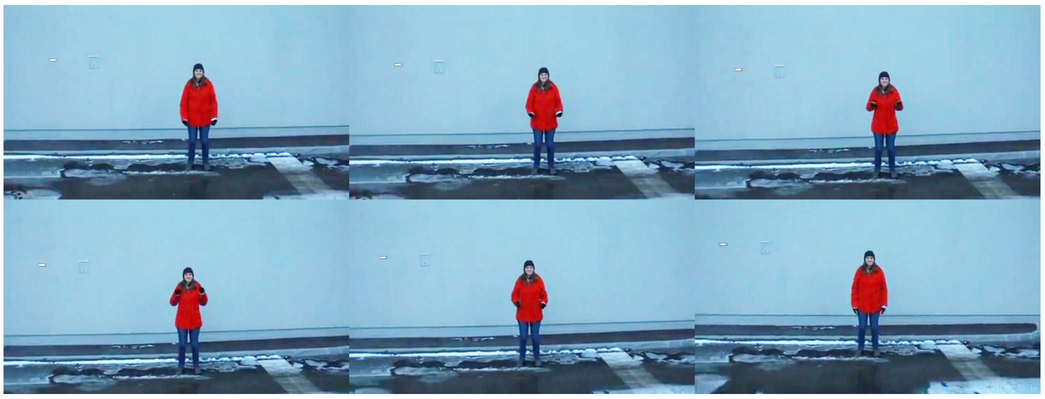


**Figure 6.7:** An example of **intuitive** gesture **up** from the dataset DS2017. This example features one of the more simple backgrounds as well as equal illumination on the scene, with the person easily distinguishable from the background. This is an example of unsteady camera. The gesture starts in a neutral position and continues into both arms being lifted in a in front of the body simultaneously, reaching the shoulder height, Then lowering back down in the same motion to the starting position.

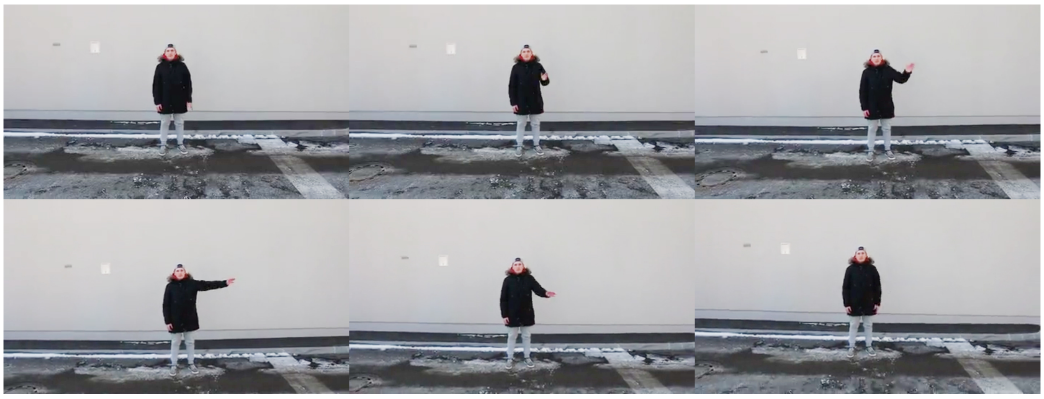


**Figure 6.8:** An example of **intuitive** gesture **left** from the dataset DS2017. This example features the same background as well as equal illumination on the scene, with the person's upper body easily distinguishable from the background. However, notice the difference in colors of the background from the previous gesture shown on Figure 6.7. This is again an example of unsteady camera. The gesture starts in a neutral position and continues into the left arm being lifted in a half circular motion to the left, as if the person is trying to push the drone away to the left. The arm is then lowered alongside the body.

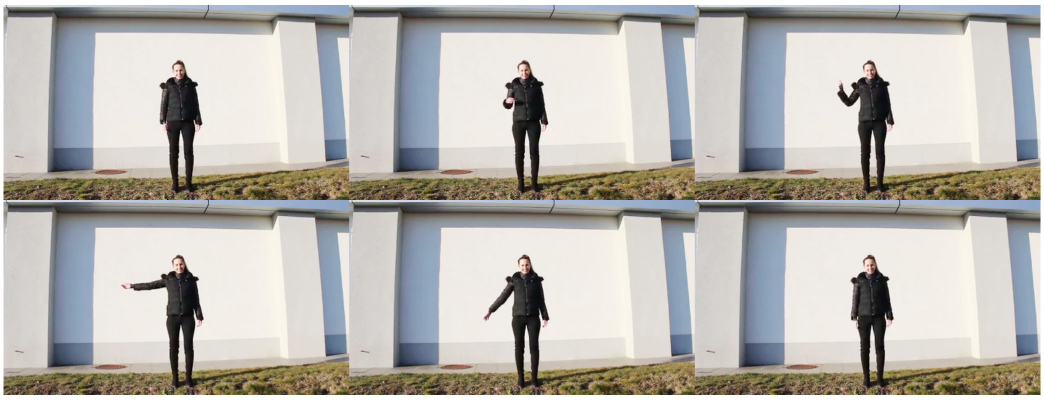


**Figure 6.9:** An example of **intuitive** gesture **right** from the dataset DS2017. This is an example of steady camera. The gesture starts in a neutral position and continues into the right arm being lifted in a half circular motion to the right, as if the person is trying to push the drone away to the right. The arm is then lowered alongside the body.

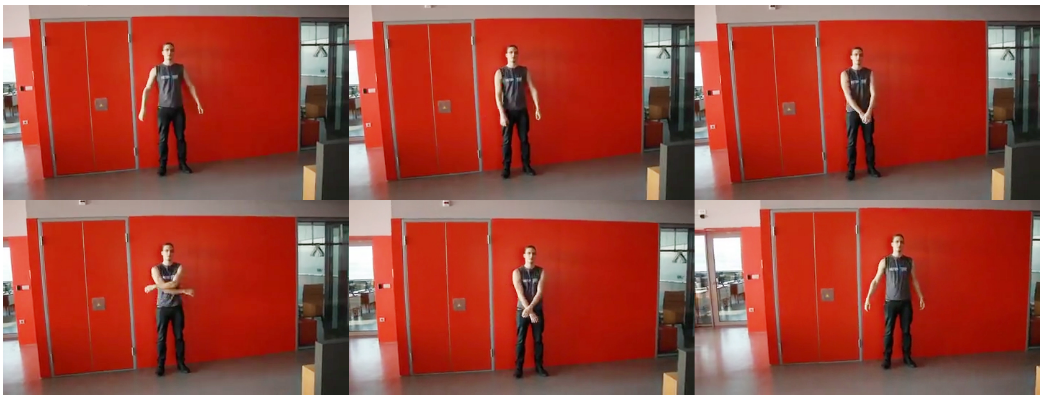


**Figure 6.10:** An example of **intuitive** gesture **down** from the dataset DS2017. This is an example of unsteady camera. The gesture starts in a neutral position and continues into both arms being criss-crossed in fron of the person's body, as if the person is trying to signal the drone to stop. The arms are then returned to neutral position.s

person. This allows us to evaluate the algorithms without the optical flow being generated by the background movements as well as keep the person at the same absolute position in the video. This sub-set combined with a controlled gesture set represents the "easiest", baseline evaluation set.

**Unsteady dataset**

Unsteady videos were taken without any stabilization and random movements of the camera were introduced. These movements are meant to simulate a flying drone, with issues with stabilization or influence from the windy environment. Due to the movements the person is not always at the same absolute position in the video, making it harder for the tracker to keep track of the person. In even worse cases, the person is sometimes slightly cropped out from the video. Due to natural light from the environments or artificial lighting in the scene, sometimes there are lens flares apparent on the video and the illumination can change slightly. This sub-set combined with the intuitive category represents the toughest part of the dataset.

| DS2017 Gesture times | |
|---|---|
| **Con/Int Gesture** | **Time [s]** |
| Control Up | 1.83 |
| Control Down | 1.82 |
| Control Left | 2.04 |
| Control Right | 1.84 |
| Intuitive Up | 1.64 |
| Intuitive Down | 1.77 |
| Intuitive Left | 1.78 |
| Intuitive Right | 1.59 |
| Average | **1.78** |

**Table 6.2:** Measured duration of each gesture in the DS2017 dataset.

### 6.2.3 Gesture duration

We measured the average gesture duration in order to decide on the correct number of frames to process for each. The measured times are presented in Table 6.2. We found that an average gesture takes 1.78 seconds or 26.7 frames when the FPS is lowered to 15.

### 6.2.4 Conclusion

For the purpose of development and evaluation of the system, we used two datasets - JHMDB and DS2017. The former consists of clips of actions taken ″in-the-wild″ with low-level and high-level features provided as annotation, allowing us to evaluate gesture recognition. The later is a dataset created for the purpose of developing our system specifically, consisting of our defined actions (up, down, left and right). It is annotated for the purpose of action detection, pose estimation and gesture recognition. DS2017 is split into 6 categories, where 2 main categories are a controlled and an intuitive set of videos, both further sub-set into a steady and unsteady dataset.

Both datasets proved to be invaluable in the development of the system, training of the models, that would detect actions and recognize gestures, and they provide a good basis for evaluation.

## 6.3 Action Detection Evaluation

### 6.3.1 Evaluation procedure

We evaluate the action detection part of the system on the *DS2017* dataset. Action detection was developed with certain restrictions in mind and DS2017 was assembled according to those. For example, we want the person to be visible in full in the frame, as we are focused on full-body gestures. This restriction is violated in most cases in JHMDB videos. DS2017 also offers an advantage, since we can evaluate various situations. Evaluations will therefore be split into

1 controlled gestures with steady camera,
2 controlled gestures with unsteady camera,
3 intuitive gestures with steady camera,
4 intuitive gestures with unsteady camera,
5 controlled gestures (combined),
6 intuitive gestures (combined).

Furthermore DS2017 enables us to evaluate action detection with respect to each gesture separately, according to the above categories. A summary of the number of gestures can be found in Table 6.1.

For each category we evaluate in how many videos the action was successfully detected and discuss the reasons why some actions are not detected. At this stage we have no information about the specific gesture that is taking place. Therefore we consider action detection to be successful when the action was detected on enough chunks, according to the threshold of the circular buffer, regardless of what specific action is in progress. For example, when we have a Circular Classification Buffer of size 6 we can specify the threshold for action to be 4 chunks, which means that action has to be detected on 4 out of 6 chunks or on 20 frames (each chunk consists of 5 frames). When action is detected (the threshold of chunks containing an action is exceeded), the frames that contain an action are forwarded to the human pose estimation model.

We will compare the effect of steady and unsteady camera as well as directly compare the detection rate for controlled gestures and intuitive gestures. For each gesture we can then determine if it is harder to detect than the other and in what circumstances that is the case.

### 6.3.2 Evaluation of controlled gestures

For controlled gestures there are a total of **320** videos in DS2017. We use a Circular Classification Buffer of size 6, where the threshold for a detected action is 4 chunks out of 6 classified as an action. How classification buffers are used is described in Section 4.2 about Action Detection. Since each chunk is of length 5 frames, we assume the controlled gesture set to have an average length of 30 frames (with FPS lowered to 15, producing 2 seconds of video). With such assumption we are able to detect the action on **275 videos or in 85.93%** of all videos having a controlled gesture. During the evaluations we are able to immediately deduce that the assumed action length has a big impact on the choice of the length of the buffer and it's threshold. For example, if we decrease the buffer size to 5 and set the limit to 3, then we would be able to detect actions on 97.18% of videos. But in the case of controlled gestures buffer size of 5 is not enough and we would cut off the last part of the action, which is sent to the pose estimation module. *Therefore, we decided to keep the buffer size at 6, with a limit of 4, for controlled gestures.*

| **Controlled** gestures detection rate | | | | |
|---|---|---|---|---|
| **Run** | Left (%) | Right (%) | Up (%) | Down (%) |
| 1 | 88.75 | 87.50 | 85 | 80 |
| 2 | 100 | 95 | 97.50 | 93.75 |
| **Intuitive** gestures detection rate | | | | |
| 1 | 45 | 52.50 | 20 | 57.50 |
| 2 | 82.50 | 83.75 | 55 | 87.50 |

**Table 6.3:** Action detection rate per gesture (left, right, up, down) for combined (steady, unsteady) **controlled** compared to **intuitive** gesture set. For *Run 1* we used buffer size of 6 and a positive chunk limit of 4, meaning that at least 4 chunks in the buffer had to be classified as containing an action, to detect the gesture. For *Run 2* we used a buffer of size 5 and a positive chunk limit of 3.

**Specific controlled gesture evaluation**

For the controlled gesture set (combining both steady and unsteady video), we show action detection rates per gesture (left, right, up, down) in Table 6.3. We show the result for buffer length of 6 and chunk limit of 4. We also show results for buffer length of 5 and chunk limit of 3, in order to compare with detection rates directly with intuitive gestures, where these parameters were chosen as the best choice.

We found that we achieve the highest accuracy in detecting gesture *right* and the lowest for gesture *down* for controlled gestures (when using buffer length 6 and positive chunk limit of 4), however the results do not vary too much for all four.

### 6.3.3 Evaluation of intuitive gestures

In comparison to controlled gestures, the algorithm struggles with the same buffer size of 6 and limit 4 on intuitive gestures, since they are generally shorter. We detect actions on only *142 videos or 44.43%* if we use a too high threshold, in our example 4. However there is one other important difference. Since the intuitive gestures are not as exuberant as controlled gestures, we noticed there are more chunks, that are in fact a part of action, classified as no action. This happens due to the particular movement of the intuitive gesture, where only small parts of the body move and produce less optical flow points if any at all. *Decreasing the buffer length to 5 with a threshold of 3 chunks exhibiting an action, is in the case of intuitive gestures a much better choice*. Since these actions are shorter (or faster), we do not loose any part of the action if the buffer is shorter. In this case we achieve **a rate of 78.12% detected actions**.

**Specific intuitive gesture evaluation**

For the intuitive gesture set (combining both steady and unsteady video), we show action detection rates per gesture (left, right, up, down) in Table 6.3. We show the result for buffer length of 6 and chunk limit of 4, as well as buffer length of 5 and chunk limit of 3, in order to compare with controlled gestures. We found that we achieve the highest accuracy in detecting gesture *down* and the lowest for gesture *up* for intuitive gestures (when using buffer length 5 and positive chunk limit of 4). Clearly the worst performing gesture is *up*. We believe the reason for this is that the gesture is usually performed in front of the person's body, which is an extremely contained area on the frame. It is also usually hard to distinguish person's arms from the torso, due to clothing (usually the sleeves have the same color as the torso), which leads to smaller numbers of optical flow points being produced.

### 6.3.4 Steady camera compared to unsteady camera

We now compare the effect of drone's movements on the scene on the action detection algorithm. Such movements can occur due to errors in stabilization or environmental effects (eg. wind). DS2017 includes videos with

| Controlled gestures detection rate | | | | |
|---|---|---|---|---|
| **Controlled** | Left (%) | Right (%) | Up (%) | Down (%) |
| Steady | 85 | 80 | 80 | 75 |
| Unsteady | 92.25 | 95 | 90 | 85 |

**Table 6.4:** Action detection rate per gesture (left, right, up, down) for **controlled and steady** compared to **controlled and unsteady** gesture set. We used buffer size of 6 and a positive chunk limit of 4.

steady camera as well as unsteady camera, simulating drone movements, which allows for a direct comparison to be made.

### Steady and unsteady camera for controlled gestures

For **steady** controlled gestures we achieve a **80.62%** detection rate, when using buffer of length 6 and chunk limit of 4. For **unsteady** videos we achieve **90%** detection rate. This may seem as a surprise, since we would expect that problems with stabilization would cause issues, however the result is perfectly reasonable. Since there are movements of the drone present in combination with person's movement, while performing gestures, the apparent motion is bigger, which produces more optical flow points on the frame thus making it easier for the action detection algorithm to pick up the ongoing action. Controlled gesture detection rate is presented in detail in Table 6.4 for steady camera videos compared to unsteady camera videos. We can see that the hardest gesture to detect in these cases is *steady down* and the easiest one to detect is *unsteady right*. We attribute the issues of detecting *steady down* to a similar cause as is with the intuitive gestures. This action requires the person to criss-cross both arms in front of the person's legs, reducing the distinction between arms and legs and in turn producing less optical flow points.

| Intuitive gestures detection rate | | | | |
|---|---|---|---|---|
| **Intuitive** | Left (%) | Right (%) | Up (%) | Down (%) |
| Steady | 80 | 80 | 42.50 | 85 |
| Unsteady | 87.50 | 87.50 | 67.50 | 92.50 |

**Table 6.5:** Action detection rate per gesture (left, right, up, down) for **intuitive and steady** compared to **intuitive and unsteady** gesture set. We used a buffer of size 5 and positive chunk limit of 3.

**Steady and unsteady camera for intuitive gestures**

For **steady** intuitive gestures we achieve a **71.87%** detection rate, when using buffer of length 5 and chunk limit of 3. For **unsteady** videos we achieve **83.75%** detection rate. As with controlled gestures the unsteady videos produce a better detection rate. Again, due to movements of the drone present in combination with person's movement, more optical flow points on the frame are produced, making it easier for the action detection algorithm to pick up the ongoing action. Intuitive gesture's detection rate is presented in detail in Table 6.5 for steady camera videos compared to unsteady camera videos. We see that in general unsteady videos produce higher accuracy in action detection. The worst performing gesture detection in both cases is again *up*, while the best performing is *down*. This possibly comes from the fact that the intuitive *down* is the most elaborate of the set, despite being similar to the *controlled down*, where it performs the worst.

### 6.3.5 Conclusion

We evaluated the action detection algorithm on various categories of the DS2017 dataset. We learned that action detection performs best on the *controlled gesture* set, as expected, since the gestures are more elaborate and standardized from person to person. We are able to detect an action in **85.93%** of cases. We learned that the *intuitive gesture* set is harder for action detection, with a detection rate of **78.12%**. Action detection performs better when the drone is *not stabilized* due to more optical flow points being generated (more apparent motion). We are also able to select Circular Classification Buffer size of 6 for controlled gestures and 5 for intuitive gestures, to deliver optimal performance, while still capturing the whole gesture. The hardest gesture to detect is *intuitive up* since it is performed in front of the person's torso, making it harder to distinguish from the "background". In fact this gesture is so hard to detect that it reduces the average detection rate of the whole *intuitive gesture* set. In general the action detection algorithm performs well, being able to detect actions when they happen, as long as they are reasonably pronounced and not too fast or too slow, but rather natural and fluent.

## 6.4 Human Pose Estimation Evaluation

In this section we will review the evaluations that were done by the original authors of the method [14], in order to understand how it performs on established benchmark datasets. We will then look at examples of the method output on our DS2017 dataset and explore where it under performs and where it performs well. Since quantitative analysis would be very time consuming, due to the lack of human joints annotations in the DS2017 dataset, we present the results in a descriptive analysis of failure and success cases. We also briefly describe the runtime evaluation of the method and report results while using the method on our use case.

Authors evaluated the method described in Section 4.3 on two well known datasets, namely MPII Human Pose Dataset [53], which is popular for multi-person human pose estimation as well as single-person human pose estimation and COCO 2016 [54], a more recent dataset created for various challenges carrying the same name.

MPII is a dataset for benchmarking articulated human pose estimation that includes 25 thousand images of 40 thousand people with annotated body joints. It covers a wide variety of 410 everyday human activities like cooking, dancing, running and so on. The data was obtained ”in-the-wild” using YouTube videos and extracting frames that display an activity.

COCO 2016 Keypoint dataset is intended for the evaluation of human joints localization ”in-the-wild”, where conditions are uncontrolled. The task is to detect people and localize their body joints. COCO 2016 keypoints challenge was part of ImageNet and COCO workshop at ECCV 2016. The dataset consists of over 100 thousand people with annotated joints.

Real-time pose estimation developed by authors [14] won the COCO 2016 keypoints challenge, where they set a new benchmark. They achieved equally high results on the MPII datasets, improving previous SOTA results for multi-person human pose estimation by a significant margin.

**Results on MPII**

MPII authors use test images to evaluate performance with their own evaluation toolkit that calculates the mean Average Precision (*mAP*) in regard to body parts using a *PCKh* threshold. *PCK*, defined by [55], is a measure of localization of the body joint, using a fraction of the person’s bounding box size (based on width and height) as a threshold for considering if the joint is within the ground truth estimation. *PCKh* is a modified metric that uses a matching threshold as 50% of the head segment length instead of the bounding box fraction. In other words both metrics are relative measures of precision, where the relative distance used as a threshold is defined either by the bounding box size of the person (PCK) or the head segment length of the person (PCKh).

The method achieved 13% higher mAP (without scale search) than previous state of the art methods evaluated on the MPII dataset. Authors found the runtime performance to be 6 orders of magnitude less than competing methods, which is a significant improvement in speed. An important distinction to the other methods pointed out by the authors is also great performance when body parts start to cross-over or overlap. This is good news for our use case, since some gestures require users to criss-cross their arms. That means the PAFs, that encode position and orientation of human body parts are performing very well and are robust to overlaps. In fact the method achieved 81.6% mAP when using ground truth body part connections with joint detection and showed only a decrease to 79.4% mAP when using PAFs to associate parts. Authors also show that mAP increases monotonically with more refinement stages used in the framework.

**Results on COCO 2016**

The COCO dataset uses similar metrics to calculate average precision (*AP*). Instead of using one threshold distance, COCO calculates AP over 10 object keypoint similarity (*OKS*) thresholds. OKS is calculated given a person's scale in the frame and the distance between predicted points and ground truth points.

Authors first show the pitfalls of top-down approaches by comparing CPMs (as described in Section 4.3.1) when using a ground truth bounding box, achieving 62.7% AP, and CPMs when using a state of the art person detector (SSD [56]), where performance drops to 52.7% AP. This shows that top-down approaches are heavily dependent on the person detector. The real-time bottom-up approach however achieves 58.4% AP. COCO 2016 Keypoints challenge also reveals a drawback of the method. It performs worse when the scale of people in the image is very small. We noticed this behavior in our use case as well, when the person is too distant from the qudcopter. Other common failure cases are rare pose articulations, while humans are performing activities such as gymnastics, occluded parts and false positives on plants, statues or animals.

### 6.4.1 Runtime Evaluation

The runtime complexity of the CNN is $\mathcal{O}(1)$ and is constant disregarding the number of people. Runtime complexity of pose parsing is $\mathcal{O}(n^2)$, with $n$ being the number of people. It is however still two magnitudes lower than the processing time of the CNN inference. According to authors parsing takes 0.58$ms$ for 9 people and CNN inference takes 99.6$ms$ on a laptop version of NVIDIA GeForce 1080-GTX GPU. The method achieves a real-time performance of 8.8 fps on a video that displays 19 people at the same time.

In our own tests, on the DS2017 dataset, the method achieves 12.3 fps, on a video displaying a single person using a slightly newer (2017) and faster NVIDIA GeForce 1080-GTX Ti GPU. Using a considerately older NVIDIA GeForce 760-GTX GPU (2013), the method achieves a performance of 1.6 fps.

### 6.4.2 Examples of human pose estimation on DS2017

In general the real-time pose estimation [14] performs well across the whole dataset, with failure cases being very rare and are usually due to cluttered backgrounds, motion blur, un-common pose articulation or underexposed frames. In some examples the pose of a joint was wrongly estimated due to a very hard shadow. The movement of the drone does not seem to affect the pose estimation at all.

#### Controlled gesture set

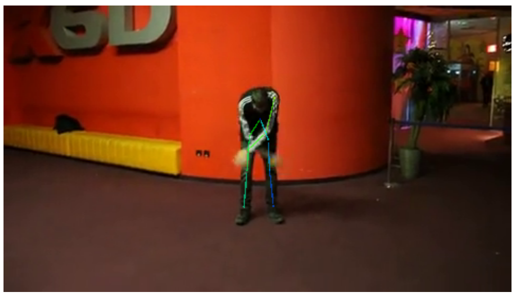
The real time pose estimation struggles the most with gesture **controlled down** . We notice that the pose is not estimated correctly when the person bends too far down, resulting in a very uncommon human pose. Such a failure case is shown on Figure 6.11a and Figure 6.11b. In that case the upper body pose is in most cases un-estimated. The second biggest problem for the pose estimation algorithm is the criss-crossing of hands during the gesture *down*. This failure case is shown on figures 6.11c, 6.11a and Figure 6.11b. We found the pose to be incorrectly estimated in almost every example of this gesture on three to five frames. However pose is only estimated incorrectly for the lower arms, while the remainder of the pose is estimated correctly. Success cases are shown on Figure 6.11d and Figure 6.11e, with the latter just a frame after criss-crossing occurs, showing that the pose estimation recovers without trouble.

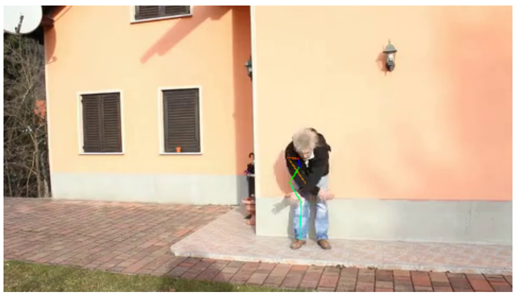
**Controlled up** has most of the human pose estimated correctly, the only issues appear when the image is too dark and the person is wearing dark clothes, so that there is little information to base the pose estimation on. Such an example is shown on Figure 6.12a. The same deficiency appears when there is too much motion blur around a particular limb (usually lower arm), in which case the joint segment for that part is not estimated. In that case the rest of the body pose is estimated correctly. Another example for a failure case for this gesture is shown on Figure 6.12b, where a part of the pose estimation is mistakenly performed on a hard shadow. Two success cases are shown on figures 6.12c and 6.12d.

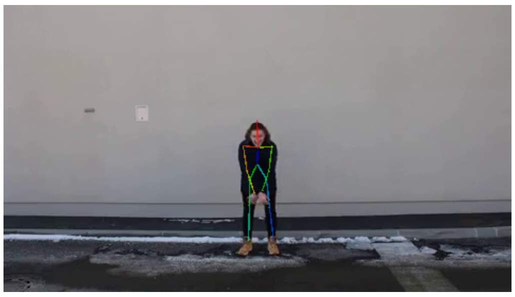
For the **controlled left** and **controlled right** gesture there is almost never a failure case. Most of them are estimated correctly. It only rarely happens that a particular limb has no pose estimation due to too much motion blur. Success cases of these gestures can be seen on figures 6.11f, 6.11g and 6.11h, where the latter two show gesture *controlled left* and the former shows gesture *controlled right*.

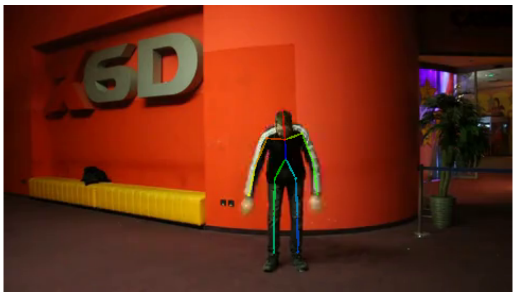
**Intuitive gesture set**

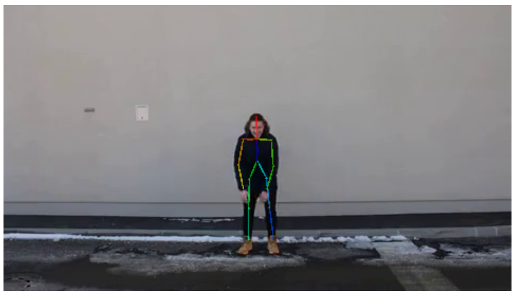
The real-time pose estimation method does not struggle as much with **intuitive down** gesture as it did with controlled down, due to the people not bending as much. The gesture is very similar to controlled down, using a criss-cross movement in front of the person's body, which still causes issues with pose estimation but for shorter periods of time (less frames). However in most cases the upper body pose is estimated correctly, due to shallower bending down of the person. We can observe that failure cases appear, when the person folds their arms towards the center of the torso, forming a square. Such cases are shown on Figure 6.13a and Figure 6.13b. We also see a failure case when a person bends down too far, as was the case in the controlled gesture set, shown on Figure 6.13c. Another failure

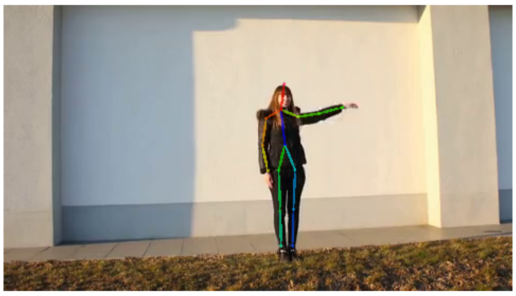

**(a)** Controlled down, fail case.

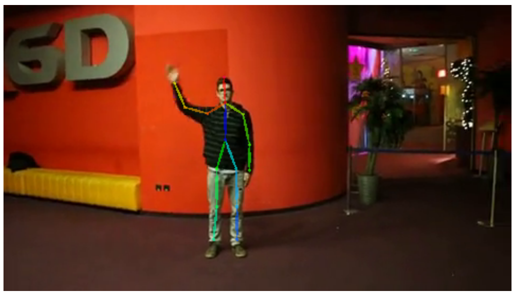

**(b)** Controlled down, fail case.


**(c)** Controlled down, fail case.


**(d)** Controlled down, success case.


**(e)** Controlled down, success case.


**(f)** Controlled left, success case.


**(g)** Controlled right, success case.

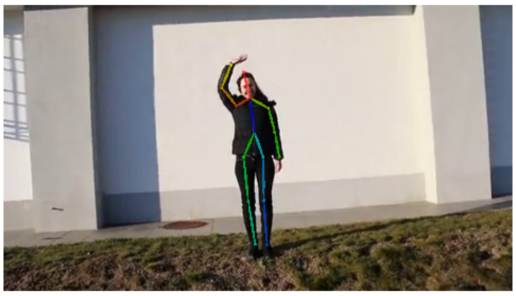
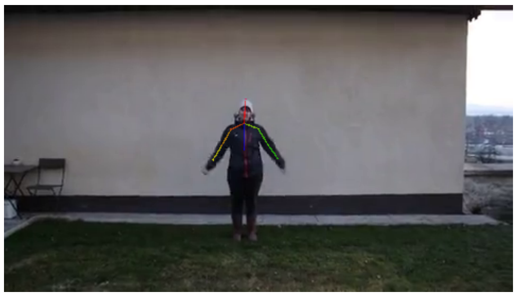
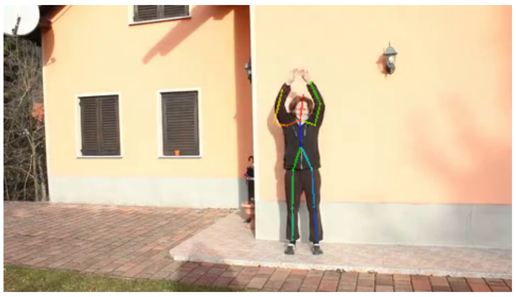
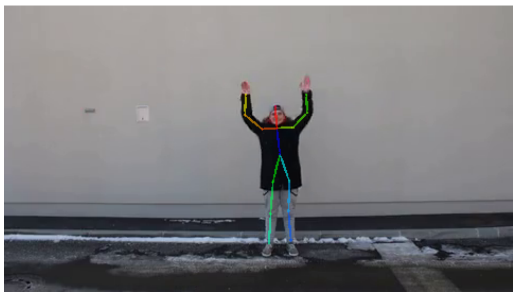

**(h)** Controlled right, success case.

**Figure 6.11:** Evaluation of the real-time pose estimation algorithm on DS2017 controlled gesture set, including both failure and success cases.


**(a)** Controlled up, fail case.


**(b)** Controlled up, fail case.


**(c)** Controlled up, success case.

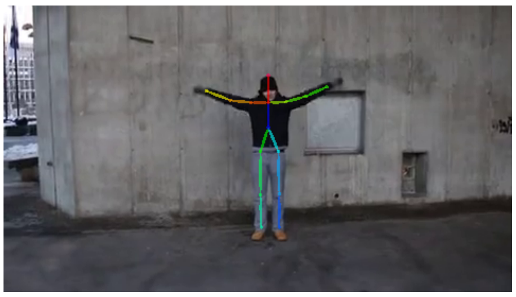

**(d)** Controlled up, success case.

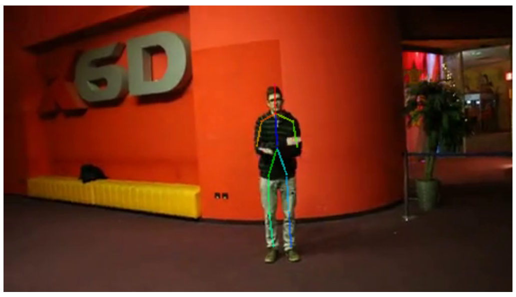
**Figure 6.12:** Evaluation of the real-time pose estimation algorithm on DS2017 Controlled gesture set including both failure and success cases.

case, as observed already before, is the hard shadow, shown on Figure 6.13d. Two success cases are shown on Figure 6.13e and Figure 6.13f, on frames directly following the criss-cross part of the gesture. Surprisingly **intuitive up** gesture is estimated correctly in most cases, despite the upper arms being completely invisible when the person raises the lower arms over them. We can see such success cases on Figure 6.13g, Figure 6.13h and Figure 6.14a. The failure cases are few and even then unrelated to the gesture itself but rather related to, eg. under exposure of the frame (shown on Figure 6.14b) or a hard shadow.

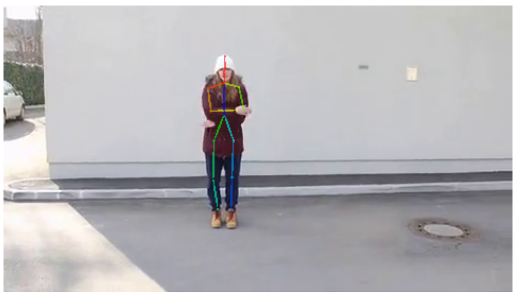
For gestures **intuitive right** and **intuitive left**, the same can be said as for the controlled left and right. The pose is estimated correctly in almost every case. Success cases are shown on figures 6.14c, 6.14d, 6.14e and Figure 6.14f. We also observe that the right gesture is a mirrored version of the left gesture for the person. In other words if the person executes a left gesture in a specific way, the right gesture will be executed in the same way. This is an observation that can not be made on the controlled gesture set, since the gestures are pre-defined. If a failure in pose estimation does occur, it is due to under exposure or a person moving their arm inwards towards the core of the body (which rarely happens). Failure cases of these two gestures are shown on Figure 6.14g and Figure 6.14h.

#### Steady camera compared to unsteady camera

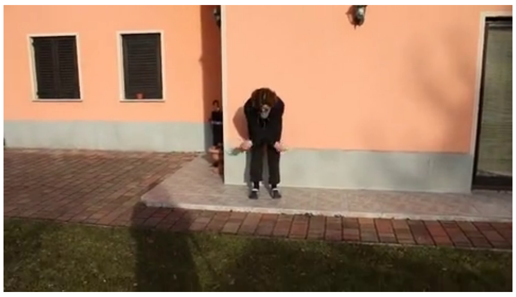
We have found there to not be a significant difference when estimating the pose on a video where the drone is stabilized compared to a moving drone. The pose estimation seems to not be affected at all, which makes sense, since it is computed frame by frame without any temporal relation. It is worth mentioning that sometimes the drone's movement partially rotates the person in view and even then the pose estimation shows no deficiencies.



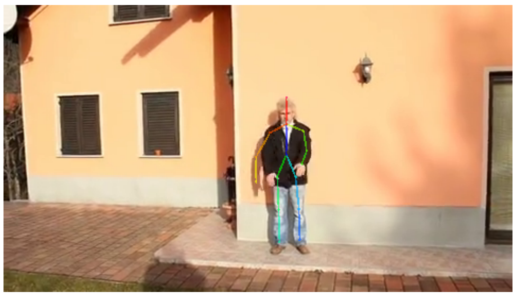
**(a)** Intuitive down, fail case.



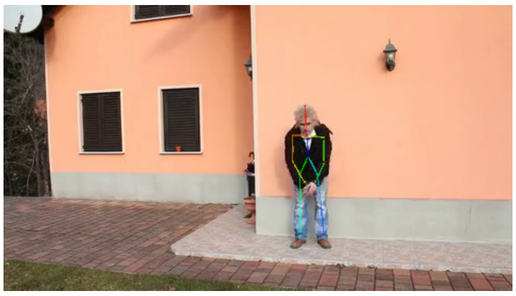
**(b)** Intuitive down, fail case.



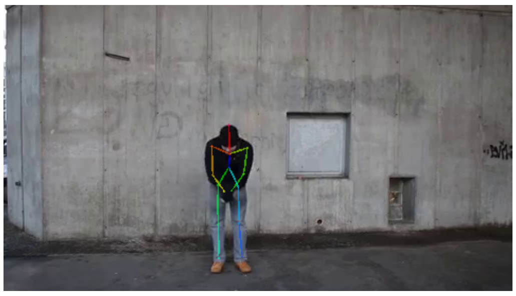
**(c)** Intuitive down, fail case.



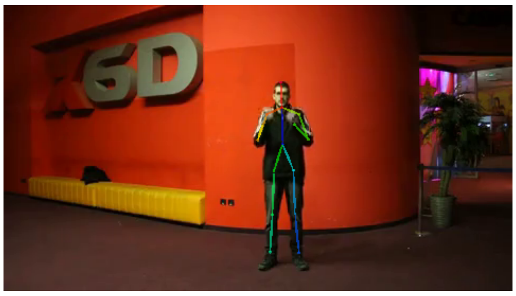
**(d)** Intuitive down, success case.



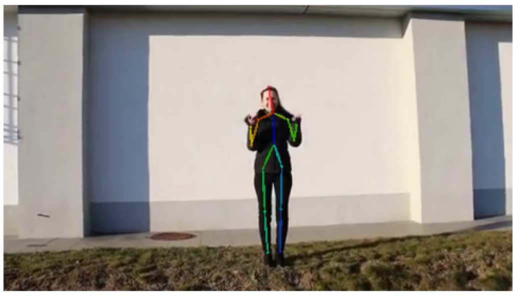
**(e)** Intuitive down, success case.



**(f)** Intuitive down, success case.

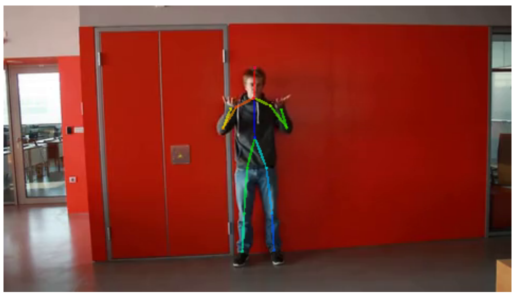


**(g)** Intuitive up, success case.

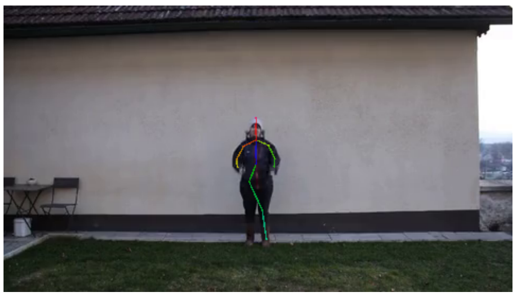


**(h)** Intuitive up, success case.

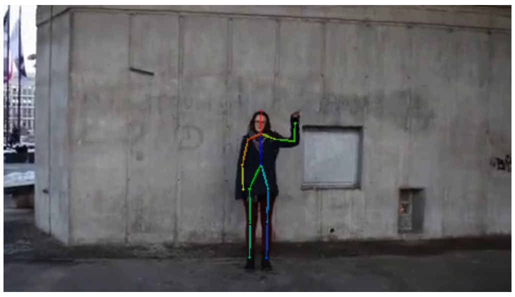
**Figure 6.13:** Evaluation of the real-time pose estimation algorithm on DS2017 Intuitive down and up gesture set, including both failure and success cases.



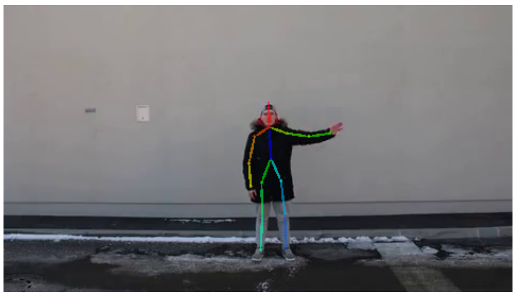
**(a)** Intuitive up, success case.



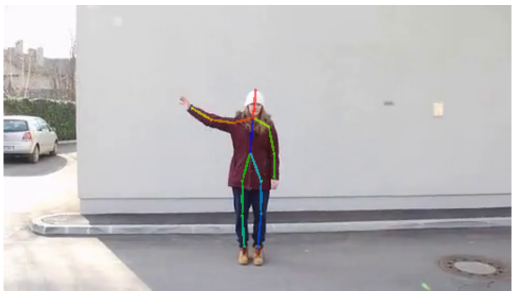
**(b)** Intuitive up, fail case.



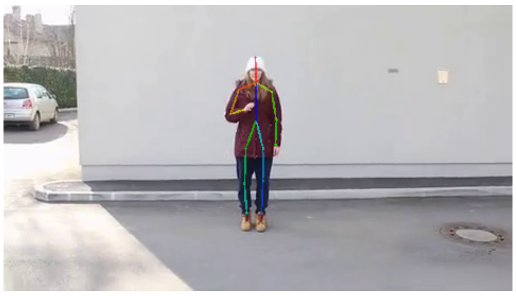
**(c)** Intuitive left, success case.



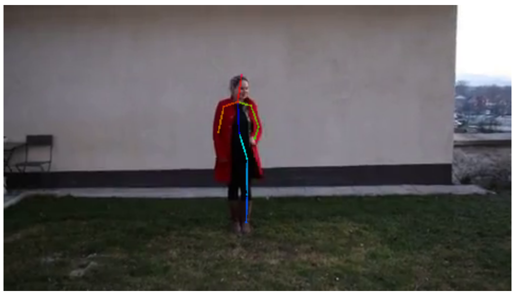
**(d)** Intuitive left, success case.



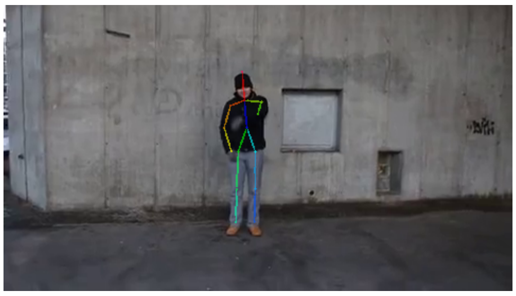
**(e)** Intuitive right, success case.



**(f)** Intuitive right, success case.



**(g)** Intuitive left, fail case.



**(h)** Intuitive left, fail case.

**Figure 6.14:** Evaluation of the real-time pose estimation algorithm on DS2017 Intuitive up, left and right gesture set, including both failure and success cases.

### 6.4.3 Conclusion

We summed up the evaluation done by authors of the real-time human pose estimation method [14] and found that the method won the COCO 2016 Keypoints challenge and set a new SOTA benchmark. It also achieved the highest results on the MPII dataset in 2016. We also discuss the runtime complexity and find that the method is indeed able to perform human pose estimation in real-time and it is 6 orders of magnitude faster than previous SOTA methods in 2016.

We evaluated the method on the DS2017 dataset and learned that most issues occur with gestures *intuitive down* and *controlled down* due to the criss-crossing of the person's hands in front of their body. The pose estimation also struggles when there are hard shadows and under exposed frames. The movement of the drone does not impact the quality of the pose estimation algorithm.

In general the pose estimation method [14] performs well, being able to estimate the pose correctly on most cases of the DS2017 gesture sets. Surprisingly it is even able to estimate the pose correctly when the lower arms completely overlap with the upper arms, as seen in the gesture *intuitive up*.

## 6.5 Gesture Classification Evaluation

### 6.5.1 Evaluation procedure

We evaluate gesture classification on both JHMDB and DS2017 datasets. Even though most actions featured in JHMDB are not really application relevant for us, it is still a good dataset for evaluating various action and gesture classification algorithms. JHMDB dataset comes annotated with action labels. It contains 21 actions in 923 videos. The videos come from YouTube and are considered to be captured "in-the-wild", which makes it an extremely difficult dataset. We evaluate the overall classification accuracy and show a confusion matrix, that shows classification accuracy for each action and for what it was mistaken, if it was not predicted correctly. We then evaluate gesture classification on all categories of the DS2017 dataset, as listed in Section 6.2. We provide the overall classification accuracy for all videos (disregarding the categories), classification accuracy and confusion matrices per category.

### 6.5.2 Evaluation on JHMDB

In general Pose Features and Bag of Words approach, as described in Section 4.4, have good performance on the JHMDB set, despite their simplicity. It is worth to note that we only use high-level features (the human pose estimation) and disregard the low-level features such as optical flow for gesture prediction.

On Figure 6.15, we show a confusion matrix for all 21 actions that make up the JHMDB dataset. We can observe that we achieve good results on some actions that involve the whole person body movement and when the person is visible in full, such as ″golf″ and ″swing baseball″. This makes sense since pose estimation features are most prominent in such cases. We also see that we achieve low classification accuracy with actions that require little movement from the person and where person is not shown in full on the video, such as ″brush hair″ and ″push″. We also see low classification accuracy for action ″jump″ where we have uncommon pose articulation. We achieve a classification accuracy of **57.08%** for the entire JHMDB dataset, using the official splits on training and testing sets. For comparison, state of the art method [57] achieves a classification accuracy of 71.08%, but it uses a much more complex method with R-CNN classification (requiring a dedicated GPU, much like the human pose estimation method that we use, and it is slower at more than 220 ms per image). We show that the Pose Features combined with Bag Of Words method is good enough for our use case, where the scene is more controlled, actions are fewer and the gestures are more exuberant. This method is also extremely fast.

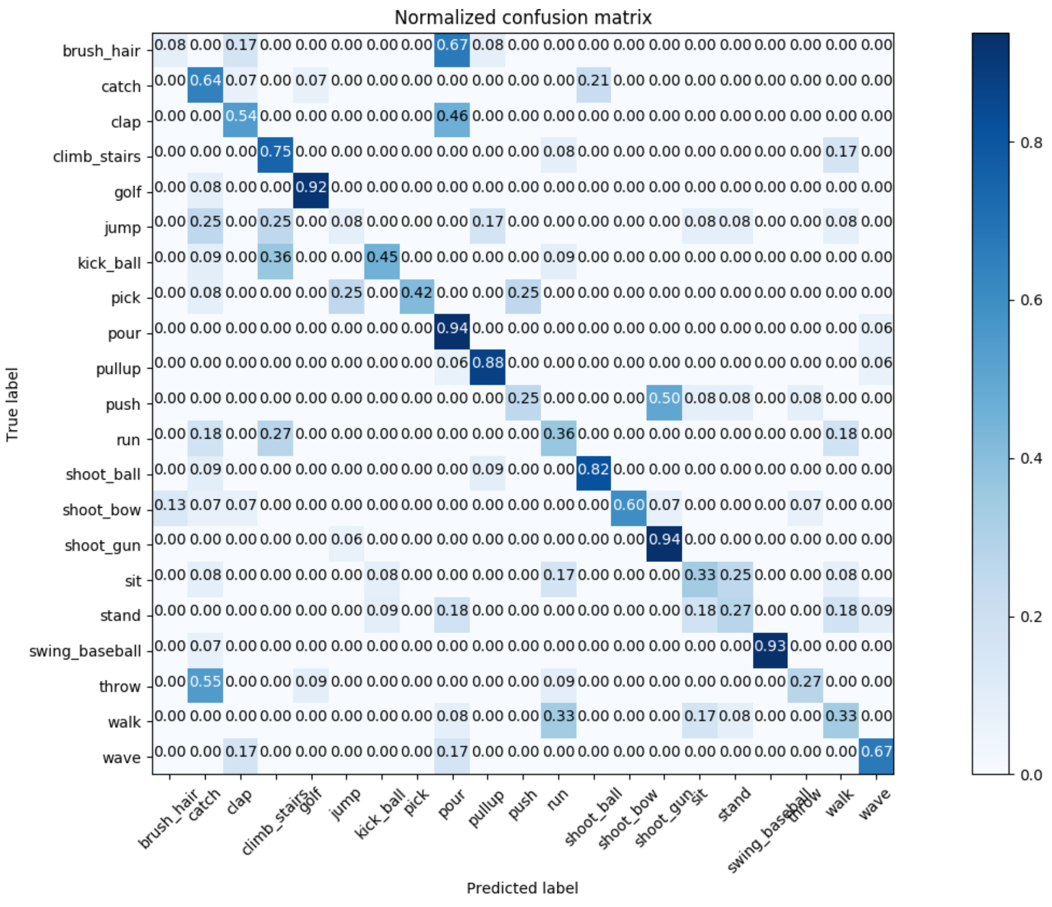


**Figure 6.15:** Confusion matrix, showing classification accuracy (CA) per class of the JHMDB Dataset.

| DS2017 classification accuracy | |
|---|---|
| **Dataset Category** | **CA** [%] |
| Control | 97.75 |
| Control Steady | 96.77 |
| Control Unsteady | 97 |
| Intuitive | 97.25 |
| Intuitive Steady | 96.5 |
| Intuitive Unsteady | 96.4 |

**Table 6.6:** Classification accuracy (CA) for the DS2017 dataset, shown for each category.

### 6.5.3 Evaluation on DS2017

We show that for our use case Pose Features and Bag of Words approach of classification works well on the dataset DS2017. Classification accuracies for each category of the dataset are shown in Table 6.6. For each category the classification accuracy is calculated as an average of the CA obtained in four different splits to training and testing sets, where 70% of examples was used for training and 30% for testing. For categories *Control* and *Intuitive* videos captured with steady camera and unsteady camera were combined to form larger training and testing sets.

#### Analysis of the control gesture set

The highest classification accuracy is obtained for the *Control* category, where the gestures are easier, more controlled and exuberant. A confusion matrix for this category is shown on Figure 6.16a. Gesture classification is 100% correct for gestures *up* and *left*. In Section 6.4, we found that the pose estimation struggled with the gesture *down* due to criss-crossing of arms and we can see that this leads to worse classification accuracy for gesture *down*. The same can be observed on confusion matrices for categories

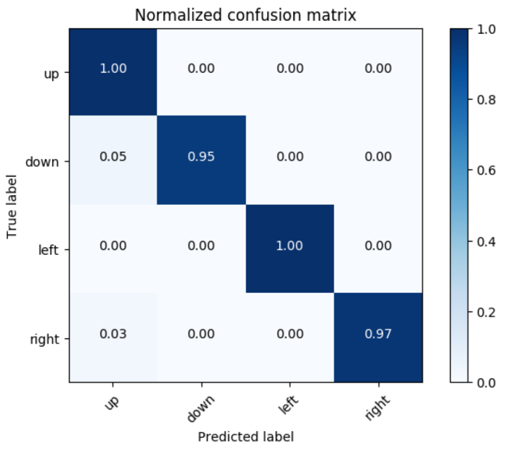
*Control Steady* (shown on Figure 6.16b) and *Control Unsteady* (shown on Figure 6.16c). In accordance to the pose estimation evaluation, we find there to be little difference for steady and unsteady sets in terms of classification accuracy.

**Analysis of the intuitive gesture set**

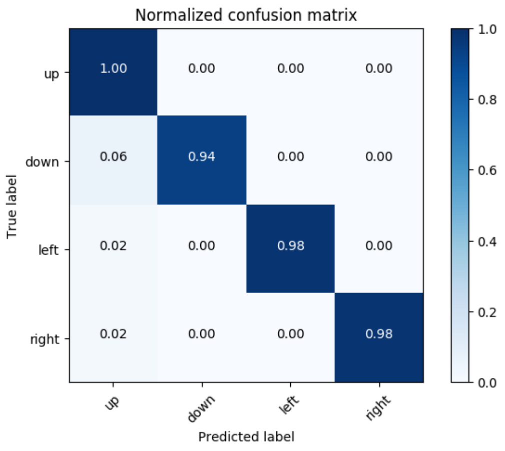
The intuitive gesture set also gets a high classification accuracy at **97.25%**, lagging just slightly behind the control gesture set. We can see the confusion matrix for category *Intuitive* on Figure 6.16d. We find that classification accuracy for gesture *up* is 100%, while with gestures *down* and *right* we achieve slightly lower classification accuracy. These results show that the classification model is also able to deal with more complex gestures, where the gestures are not as exuberant, providing the algorithm with less information based just on Pose Features. Similarly both *Intuitive Steady* (confusion matrix shown on Figure 6.16e) and *Intuitive Unsteady* (shown on Figure 6.16f) categories achieve high accuracies, with gesture *down* performing slightly worse. A slight drop in classification accuracy from the combined category *Intuitive* can perhaps be due to less training data in the set.

### 6.5.4 Runtime evaluation

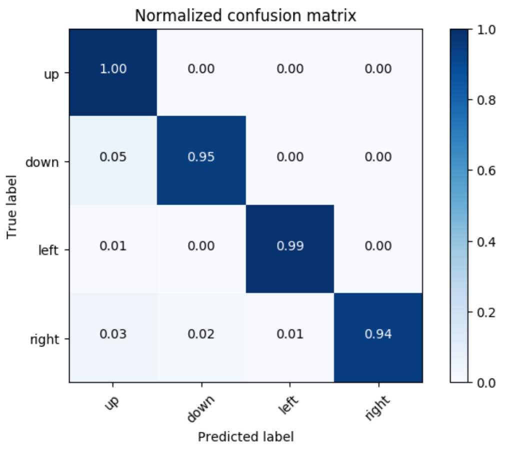
Pose Features are extremely fast to compute, and take only *503 microseconds* per frame on average. That results in *20.57 milliseconds* per an *entire gesture video*. To compute the histograms given the k-means clusters, the algorithm takes *34 milliseconds*, when using 20 clusters (the same number used for reporting the above classification accuracies) per video. This time increases with the number of clusters, since distance to each has to be computed for each frame. For example, it would take on average 280 milliseconds for 200 clusters, but the classification accuracy doesn't increase further well before that number of clusters. The final SVM prediction takes *24.6 microseconds on average*. The reported numbers were measured on an Intel Core i7 running



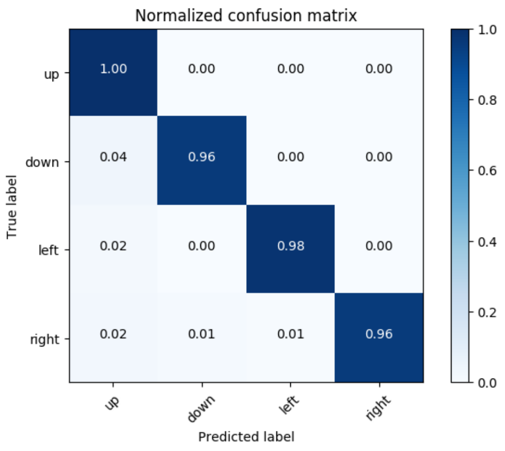
**(a)** DS2017 Control



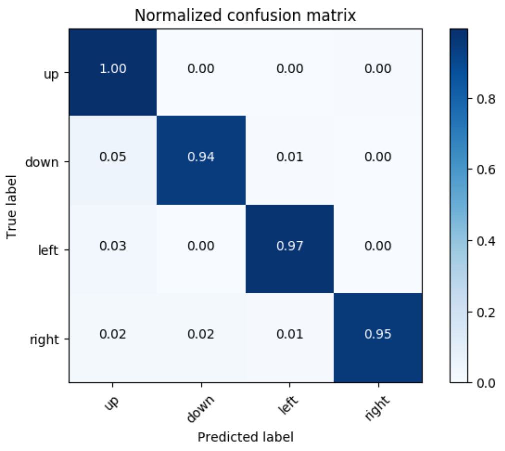
**(b)** DS2017 Control Steady



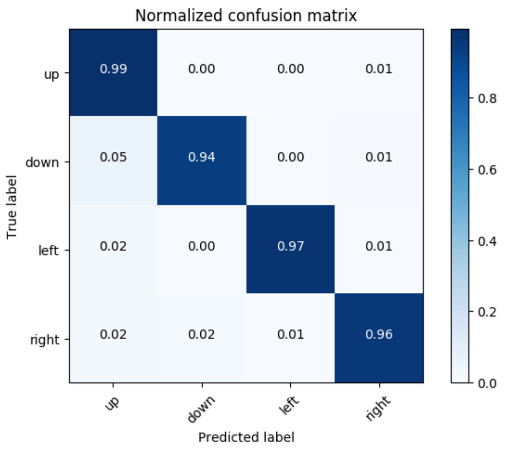
**(c)** DS2017 Control Unsteady



**(d)** DS2017 Intuitive



**(e)** DS2017 Intuitive Steady



**(f)** DS2017 Intuitive Unsteady

**Figure 6.16:** Confusion matrices for the evaluation of gesture classification algorithm on DS2017 categories.

at 2.9 GHz.

In total, the average running time *per gesture* is therefore **55 milliseconds**[1].

In summary Pose Features are extremely fast to compute and so is the Bag of Words classification, providing us real-time performance.

## 6.6 Integrated System Evaluation

### 6.6.1 Evaluation procedure

For evaluating the whole integrated system we use the DS2017 dataset. We are interested in the classification accuracy after the video is sent through the whole pipeline. First the video is sent through the action detection algorithm to forward frames and human pose estimation is performed on these frames afterwards. Pose Features are then computed and the final prediction is given by the SVM. For the purpose of this evaluation, we have trained our model on both types of gesture sets, *control* and *intuitive*.

### 6.6.2 Evaluation on DS2017

As before we evaluate the gesture sets separated on *steady* and *unsteady* sets. We only classify the videos that were left out from the training set, adhering to the splits to training (75% of examples) and testing (25% of examples) sets, as we did in the standalone component evaluation.

The action detection algorithm detected the action in **83,13%** of cases, or on 133 out of 160 test gesture videos in total. On the videos that had no action detected there are no action containing frames forwarded to the human pose estimation module and therefore the gestures are not classified.

First we present gesture classification accuracy with undetected gesture rate taken into account, to show the complete system performance. In other

[1]Note that the state of the art method [57] on JHMDB takes 220 ms, or more, for a single frame. This would result in **5 seconds**, or longer, for classifying an entire gesture video.

words, if the action detection module failed to detect an on-going action, it is treated as miss-classification. For each category the classification accuracy is calculated and presented in a confusion matrix, which also includes undetected gestures, on Figure 6.17.

Then we focus on the videos that had their action successfully detected. Observing the forwarded frames hinted, that action detection in some cases mistakenly does not detect action on the first chunk (first 5 frames), and so the action is slightly cut off in the beginning. It is rarely cut off in the end, due to the fixed buffer length. However this did not seem to have a big impact on the entire system performance.

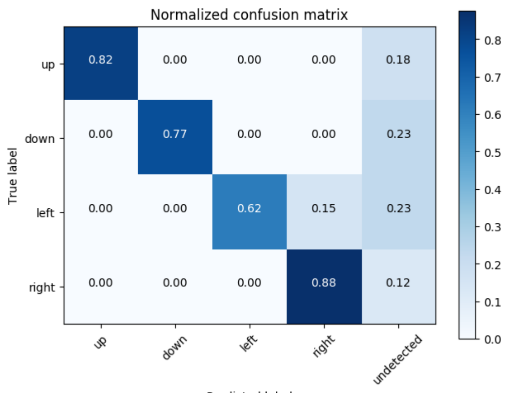
For each category the classification accuracy is calculated and presented in a confusion matrix on Figure 6.18.

**Analysis of gesture sets**

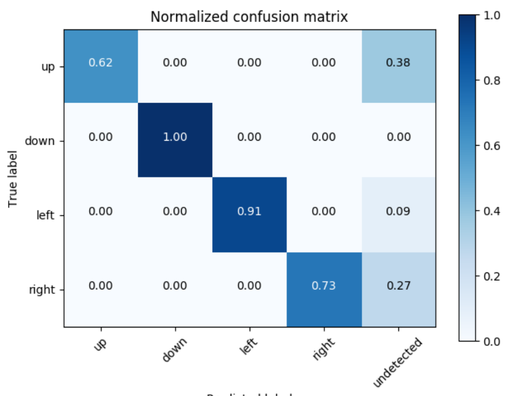
The highest overall accuracy is achieved on the *Controlled unsteady* gesture set, with an overall accuracy of 81%, as shown on Figure 6.17b. The lowest classification accuracy is achieved on *Intuitive unsteady* gesture set, as shown on Figure 6.17d, with an overall accuracy of 73%. The results confirm that the action detection module is struggling with detecting gesture *up*, which decreases the classification accuracy in all categories, except for category *Control steady*, where gesture *right* has the lowest accuracy. The overall accuracy achieved for *Intuitive steady* gesture set is 76%, shown on Figure 6.17c. Overall accuracy for *Controlled steady* gesture set is 77%, shown on Figure 6.17a.

**Analysis of detected gesture sets**

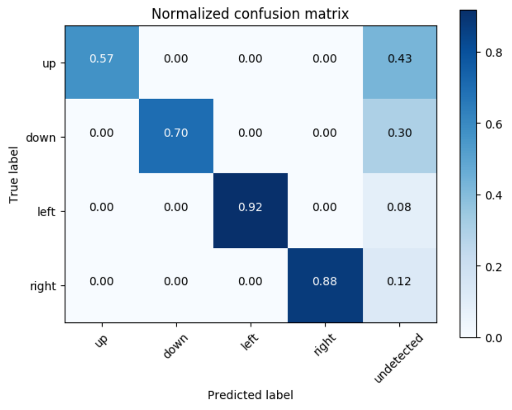
The classification accuracy is similar for both *Controlled* and *Intuitive* categories. We achieve **100%** accuracy in all categories for gestures *up, down* and *right*, while the gesture *left* is in some cases mistaken for gesture *right*. This could be due to their similarity. It seems that if action detection successfully detects the action and forwards the gesture to pose estimation, the gestures are correctly classified. We observed lower classification ac-



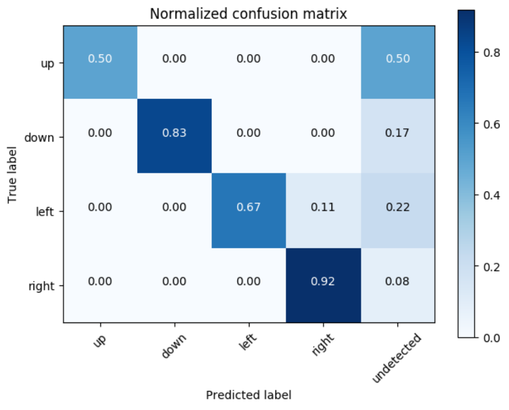
**(a)** DS2017 Control Steady



**(b)** DS2017 Control Unsteady



**(c)** DS2017 Intuitive Steady



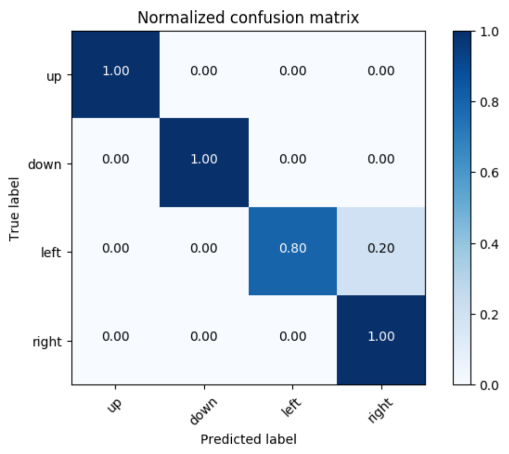
**(d)** DS2017 Intuitive Unsteady

**Figure 6.17:** Confusion matrices for the evaluation of the gesture control system on DS2017 categories, which include the undetected gesture rates. If the gesture was not detected with the action detection module, it is treated as miss-classified and presented as category "undetected".

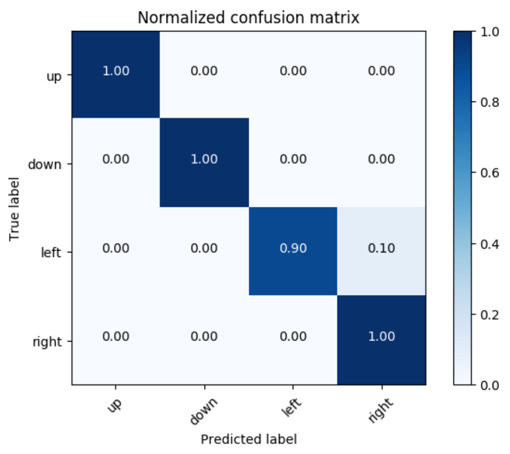
curacies before in the standalone evaluation of the gesture classification in Section 6.5, which perhaps tells us that the gestures that are not detected in the action detection part are irregular in terms of exuberance (not enough movement or restricted movement) or not salient enough from the background.

### 6.6.3 Runtime evaluation

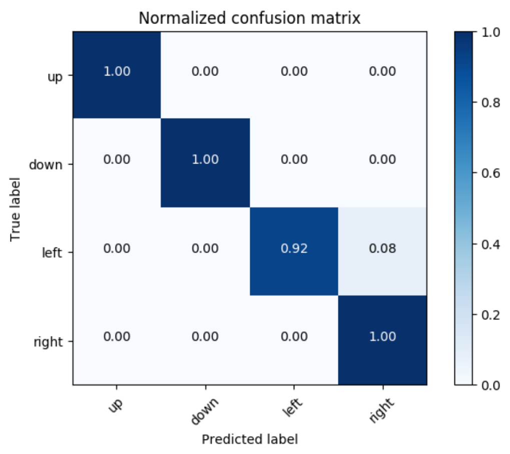
Action detection is performing in real-time with an average of *58 milliseconds* per frame. The pose estimation takes on average *2,03 seconds* to estimate the pose for 25 frames (the entire gesture sent forward by action detection). Gesture classification takes *54 milliseconds on average*. From the detected action to the final gesture prediction it takes on average **2,14 seconds**. Time was measured on a PC with a Core i7 4770K, running at 3.5 GHz. The whole pipeline would run slower on a quadcopter, due to a less powerful CPU. Despite that, most time is spent for pose estimation, which is run on an external PC and the total time would not be much longer.



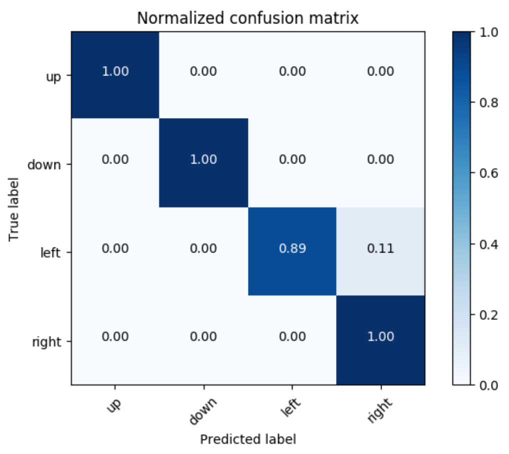
**(a)** DS2017 Control Steady



**(b)** DS2017 Control Unsteady



**(c)** DS2017 Intuitive Steady



**(d)** DS2017 Intuitive Unsteady

**Figure 6.18:** Confusion matrices for the evaluation of the gesture control system on DS2017 categories, given a detected action by the action detection module.

# Chapter 7

# Conclusion and future work

## 7.1 Concluding remarks

In this work we set out to develop a real-time gesture control system for quadcopters. We learned about the previous research done on the matters and how it was usually done with external devices, such as depth sensors and motion control sensors.

We assembled our own mobile platform using "do-it-yourself" components and spare parts with an advanced open source autopilot and a powerful on-board PC.

We developed a three-phase gesture control system, by first detecting when the action is happening on the video, with the help of a fast person detector, a fast tracker and optical flow. After detecting the action, we use a state of the art method [14] to estimate the human pose on the frames that contain an action. The method uses advanced convolutional neural networks and is able to achieve real-time performance by combining human pose estimation with part association in the inference stage. After getting the locations of joints in the human pose, we compute relational Pose Features, that provide features for SVM classification, which predicts the final gesture.

We implemented the integrated gesture control system with the open source library OpenCV and meta operating system ROS, providing us with the modular distributed design that alows us to run the most demanding part of the system - human pose estimation - on an external PC, due to strict GPU requirements. It also makes our system ready for full on-line deployment, when the hardware meets the requirements. We also designed a circular buffer action detection system allowing us to operate on on-line always incoming video.

For the purpose of development and evaluation we assembled our own dataset DS2017 that features 640 gestures performed by 20 people. DS2017 is split into two major categories. The first includes controlled gestures up, down, left and right and the second includes intuitive gestures up, down, left and right. Each category is further split into a stabilized camera set and an unstabilized camera set, to simulate drone movements due to errors in stabilization, or environmental influence.

Evaluations of our algorithms show that we are able to detect actions in 83% of cases, having more success with the controlled gestures, while intuitive gestures are a harder challenge. We also found that the method [14] is extremely good in estimating the human pose on our dataset, struggling only when the hands criss-cross, as seen in gestures “down”, or other factors that concern the conditions in which the videos were taken. We are also able to reach a high classification accuracy of 96.8% on our DS2017 dataset for the final gesture prediction.

Our system takes **2.14** seconds on average to send a command to the quadcopter from the time when the action was detected, which introduces a small delay but it is nevertheless still very fast.

## 7.2 Future work

The part of the system that is preventing us from achieving true real-time and on-line performance is the human pose estimation. Although the method we use is extremely fast, it still takes about 2 seconds to estimate the pose for one gesture. And to achieve this time we need a specific GPU, that is installed in an external PC.

The need for a specific external GPU could be prevented if deep learning frameworks start supporting ARM based GPUs, as well as other manufacturers, apart from NVIDIA. There has been some recent advances in that area, but more is needed to be done by the deep learning community.

Another way we could eliminate the need for an external GPU, is to use the specific required GPU on the quadcopter itself. We already discussed the *NVIDIA Jetson TK 1* in Section 3.1, which provides CUDA cores, that are required for our application. During this time NVIDIA already announced two successors, *NVIDIA Jetson TX 1* and *NVIDIA Jetson TX 2*, that includes 256 CUDA cores and is based on the same architecture as the NVIDIA 1080 GTX Ti GPU, that we used in our evaluations. This would not be enough but it is a good start. However there is still a lack of main boards on which these embedded boards can be mounted. Perhaps this will change in the near future.

Due to our implementation in ROS, we are not required to run the Pose Estimation Node on an external PC, so it would be easy to run our system on the drone completely.

To reduce the requirements in terms of numbers of CUDA cores and make the human pose estimation faster, we could try to implement frame-to-frame interpolation, so that we would estimate the pose on a select number of frames and then interpolate the estimated joints to the rest.

We would also like to add more gestures to the DS2017 dataset and support them in our gesture control system in the future.

# Bibliography

[1] K. Pfeil, S. L. Koh, J. LaViola, Exploring 3d gesture metaphors for interaction with unmanned aerial vehicles, in: Proceedings of the 2013 international conference on Intelligent user interfaces, ACM, 2013, pp. 257–266.

[2] A. Sarkar, K. A. Patel, R. G. Ram, G. K. Capoor, Gesture control of drone using a motion controller, in: Industrial Informatics and Computer Systems (CIICS), 2016 International Conference on, IEEE, 2016, pp. 1–5.

[3] You can now fly a toy drone with just hand gestures, `https://www.cnet.com/news/`, accessed: 2017-06-4.

[4] A gesture controlled drone makes the user look like a jedi, `http://www.businessinsider.com/gesture-controlled-drone-2015-12?IR=T`, accessed: 2017-06-4.

[5] J. Nagi, A. Giusti, G. A. Di Caro, L. M. Gambardella, Hri in the sky: controlling uavs using face poses and hand gestures.

[6] R. D. Geest, E. Gavves, A. Ghodrati, Z. Li, C. Snoek, T. Tuytelaars, Online action detection (2016). `arXiv:1604.06506`.

[7] G. Gkioxari, J. Malik, Finding action tubes, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 759–768.

[8] A. Yao, J. Gall, G. Fanelli, L. Van Gool, Does human action recognition benefit from pose estimation?", in: Proceedings of the 22nd British machine vision conference-BMVC 2011, 2011.

[9] M. Müller, T. Röder, M. Clausen, Efficient content-based retrieval of motion capture data, in: ACM Transactions on Graphics (TOG), Vol. 24, ACM, 2005, pp. 677–685.

[10] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, M. J. Black, Towards understanding action recognition, in: Proceedings of the IEEE International Conference on Computer Vision, 2013, pp. 3192–3199.

[11] T. Pfister, J. Charles, A. Zisserman, Flowing convnets for human pose estimation in videos, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1913–1921.

[12] L. Bourdev, J. Malik, Poselets: Body part detectors trained using 3d human pose annotations, in: Computer Vision, 2009 IEEE 12th International Conference on, IEEE, 2009, pp. 1365–1372.

[13] S.-E. Wei, V. Ramakrishna, T. Kanade, Y. Sheikh, Convolutional pose machines, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4724–4732.

[14] Z. Cao, T. Simon, S.-E. Wei, Y. Sheikh, Realtime multi-person 2d pose estimation using part affinity fields, in: CVPR, 2017.

[15] H. Cheng, L. Yang, Z. Liu, Survey on 3d hand gesture recognition, IEEE Transactions on Circuits and Systems for Video Technology 26 (9) (2016) 1659–1673.

[16] A. R. Sarkar, G. Sanyal, S. Majumder, Hand gesture recognition systems: a survey, International Journal of Computer Applications 71 (15).

[17] R. Z. Khan, N. A. Ibraheem, Survey on gesture recognition for hand image postures, Computer and Information Science 5 (3) (2012) 110.

[18] Iron man, Director: Jon Favreau, Actors: Robert Downey Jr., Gwyneth Paltrow, Terrence Howard, Publisher: Paramount Pictures, Year: 2008.

[19] J. R. Cauchard, K. Y. Zhai, J. A. Landay, et al., Drone & me: an exploration into natural human-drone interaction, in: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, ACM, 2015, pp. 361–365.

[20] E. Peshkova, M. Hitz, B. Kaufmann, Natural interaction techniques for an unmanned aerial vehicle system, IEEE Pervasive Computing 16 (1) (2017) 34–42.

[21] Maestro glove, `http://maestroglove.com`, accessed: 2017-06-4.

[22] Y. Song, D. Demirdjian, R. Davis, Tracking body and hands for gesture recognition: Natops aircraft handling signals database, in: Automatic Face & Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on, IEEE, 2011, pp. 500–506.

[23] H. Wang, C. Schmid, Action recognition with improved trajectories, in: Proceedings of the IEEE International Conference on Computer Vision, 2013, pp. 3551–3558.

[24] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556.

[25] L. Cao, Z. Liu, T. S. Huang, Cross-dataset action detection, in: Computer vision and pattern recognition (CVPR), 2010 IEEE conference on, IEEE, 2010, pp. 1998–2005.

[26] A. Yao, J. Gall, L. Van Gool, A hough transform-based voting framework for action recognition, in: Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, IEEE, 2010, pp. 2061–2068.

[27] H. Wang, A. Kläser, C. Schmid, C.-L. Liu, Action recognition by dense trajectories, in: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, IEEE, 2011, pp. 3169–3176.

[28] C. Wang, Y. Wang, A. L. Yuille, An approach to pose-based action recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 915–922.

[29] Y. Yang, D. Ramanan, Articulated pose estimation with flexible mixtures-of-parts, in: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, IEEE, 2011, pp. 1385–1392.

[30] P. Buehler, M. Everingham, D. P. Huttenlocher, A. Zisserman, Upper body detection and tracking in extended signing sequences, International journal of computer vision 95 (2) (2011) 180–197.

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

[32] G. Chéron, I. Laptev, C. Schmid, P-cnn: Pose-based cnn features for action recognition, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 3218–3226.

[33] Open hardware project pixhawk, `https://pixhawk.org`, accessed: 2017-06-26.

[34] Holybro pixfalcon fmu, `http://www.holybro.com/product/8`, accessed: 2017-06-26.

[35] Qgroundcontrol, `http://qgroundcontrol.com`, accessed: 2017-06-26.

[36] Mavlink - communication library, `https://mavlink.io/en/`, accessed: 2017-06-26.

[37] Hardkernel odroid xu 4, `http://www.hardkernel.com/main/products/prdt_info.php`, accessed: 2017-06-26.

[38] Nvidia jetson tk1 embedded developer kit, `http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html`, accessed: 2017-06-26.

[39] Qualcomm snapdragon flight kit, `https://www.intrinsyc.com/vertical-development-platforms/qualcomm-snapdragon-flight/`, accessed: 2017-06-26.

[40] S. Zhang, R. Benenson, M. Omran, J. Hosang, B. Schiele, How far are we from solving pedestrian detection?, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 1259–1267.

[41] R. Benenson, M. Mathias, R. Timofte, L. Van Gool, Pedestrian detection at 100 frames per second, in: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE, 2012, pp. 2903–2910.

[42] N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, Vol. 1, IEEE, 2005, pp. 886–893.

[43] T. Vojir, J. Noskova, J. Matas, Robust scale-adaptive mean-shift for tracking, Pattern Recognition Letters 49 (2014) 250–258.

[44] D. Comaniciu, V. Ramesh, P. Meer, Real-time tracking of non-rigid objects using mean shift, in: Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on, Vol. 2, IEEE, 2000, pp. 142–149.

[45] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, F. Porikli, L. Čehovin, G. Nebehay, G. Fernandez, T. Vojir, A. Gatt, A. Khajenezhad, A. Salahledin, A. Soltani-Farani, A. Zarezade, A. Petrosino, A. Milton, B. Bozorgtabar, B. Li, C. S. Chan, C. Heng, D. Ward, D. Kearney, D. Monekosso, H. C. Karaimer, H. R. Rabiee, J. Zhu, J. Gao, J. Xiao, J. Zhang, J. Xing, K. Huang, K. Lebeda, L. Cao, M. E. Maresca, M. K. Lim, M. E. Helw, M. Felsberg, P. Remagnino, R. Bowden, R. Goecke, R. Stolkin, S. Y. Lim, S. Maher, S. Poullot, S. Wong, S. Satoh, W. Chen,

W. Hu, X. Zhang, Y. Li, Z. Niu, The Visual Object Tracking VOT2013 challenge results (Dec 2013).

[46] M. A. Fischler, R. C. Bolles, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, Communications of the ACM 24 (6) (1981) 381–395.

[47] U. Mahbub, H. Imtiaz, M. A. R. Ahad, An optical flow based approach for action recognition, in: Computer and Information Technology (ICCIT), 2011 14th International Conference on, IEEE, 2011, pp. 646–651.

[48] V. Ramakrishna, D. Munoz, M. Hebert, J. Andrew Bagnell, Y. Sheikh, Pose Machines: Articulated Pose Estimation via Inference Machines, Springer International Publishing, Cham, 2014, pp. 33–47. doi:10.1007/978-3-319-10605-2_3.
URL https://doi.org/10.1007/978-3-319-10605-2_3

[49] D. B. West, et al., Introduction to graph theory, Vol. 2, Prentice hall Upper Saddle River, 2001.

[50] H. W. Kuhn, The hungarian method for the assignment problem, Naval Research Logistics (NRL) 2 (1-2) (1955) 83–97.

[51] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, M. J. Black, Towards understanding action recognition, in: International Conf. on Computer Vision (ICCV), 2013, pp. 3192–3199.

[52] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, arXiv preprint arXiv:1408.5093.

[53] M. Andriluka, L. Pishchulin, P. Gehler, B. Schiele, 2d human pose estimation: New benchmark and state of the art analysis, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.

[54] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, C. L. Zitnick, Microsoft coco: Common objects in context, in: European conference on computer vision, Springer, 2014, pp. 740–755.

[55] Y. Yang, D. Ramanan, Articulated human detection with flexible mixtures of parts, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (12) (2013) 2878–2890.

[56] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, A. C. Berg, Ssd: Single shot multibox detector, in: European conference on computer vision, Springer, 2016, pp. 21–37.

[57] X. Peng, C. Schmid, Multi-region two-stream r-cnn for action detection, in: European Conference on Computer Vision, Springer, 2016, pp. 744–759.