UNIVERSITY OF TARTU

Faculty of Science and Technology

Institute of Technology

Igor Rybalskii

Gesture Detection Software for Human-Robot Collaboration

Bachelor's Thesis (12 ECTS)

Curriculum Science and Technology

Supervisors:

Junior research fellow, MSc Robert Valner

Associate professor, PhD Karl Kruusamäe

Tartu 2020

Gesture Detection Software for Human-Robot Collaboration

Abstract:

With robots becoming more complex machines with more actions available at their disposal, it becomes harder for humans to control them without prior training. I propose a gesture detection system which uses OpenPose and ROS (Robot Operating System) to control mobile robotic platforms. Output from OpenPose is normalized into a joint angle form, which is also used to describe gestures in the system. Proposed normalization method in combination with the capability to change described gestures in a separate YAML configuration file makes the whole system scalable for a developer who can add, remove or modify gestures described by angle notation. The developed system is able to detect static gestures and was tested on three sets, each consisting of 5 gestures to control a Clearpath Jackal mobile robot.

Keywords: gesture detection, human-robot collaboration, OpenPose, ROS

CERCS: T120 - Systems engineering, computer technology, T121 - Signal processing, T125 - Automation, robotics, control engineering

Žestituvastus tarkvara inimese ja roboti koostööks

Lühikokkuvõte:

Robotid on muutumas tehniliselt aina keerukamaks ning nende abil on võimalik täita üha enam ülesandeid. Ka robotite juhtimine on inimestele muutumas väga keeruliseks. Käesolevas lõputöös luuakse kehakeele-põhine süsteem, mis kasutab tarkvarateeke OpenPose ja ROS, et juhtida mobiilset robotplatvormi. OpenPose'i väljund normeeritakse nurkade esitlusele, milles on kirjeldatud ka kasutatavad žestid. Loodud süsteem on skaleeritav, sest normeeritud kujul žeste saab robotsüsteemi arendaja vastavalt vajadusele lisada, muuta ja

eemaldada YAML-tüüpi konfiguratsioonifailis. Valminud lahenduse demostreerimiseks implementeeriti kolm erinevat 5-žestilist komplekti, mille abil juhiti Clearpath Jackal mobiilset robotit.

Võtmesõnad: žestituvastus, inimene ja roboti koostöö, OpenPose, ROS

CERCS: T120 - Süsteemitehnoloogia, arvutitehnoloogia, T121 - Signaalitöötlus , T125 - Automatiseerimine, robootika, juhtimistehnika

TABLE OF CONTENTS

1. ABBREVIATIONS

ANN - Artificial Neural Network

CNN - Convolutional Neural Network

COCO - Common Objects in Context

DTW - Dynamic Time Warping

HD - High Definition

HMM - Hidden Markov Model

KF - Kalman Filter

KNN - K-Nearest Neighbor

MPII - Max Planck Institute of Informatics

PF - Particle Filter

RF - Radio Frequency

RGB - Red Green Blue

ROS - Robot Operating System

SVM - Support Vector Machine

UGV - Unmanned Ground Vehicle

YAML - Yet Another Markup Language

2. INTRODUCTION

Robots are becoming increasingly complex, having different sensors, actuators, etc. Such systems require complex controller devices to have the control over every part of the robot and. These controllers are hard to use without some prior training. This can become a problem during emergency situations, where it is dangerous to send people, such as Fukushima or Chernobyl disasters. It is better to send robots, but it takes time to train the specialist in, for example, nuclear physics how to control some complex robot. More natural and fluent ways of communication can make training faster.

Alternative ways to communicate with computers and robots are being developed, such as voice recognition or gesture detection systems. Gesture detection systems are able to detect and recognize humans' gestures as commands for the robot and trigger the corresponding action. These systems have a problem, that all detected gestures are predefined and robot operators might have no control over which gestures to use. This leads to the fact that in the end operators will anyway need time to remember all control gestures.

To solve this problem and make gesture control more fluent and natural I propose a gesture detection system, which uses joint angles as a description factor for gestures and is able to change the gesture set without the change of code.

3. LITERATURE REVIEW

Controlling robots and systems alike via body and hand gestures has shown to be a viable alternative to keyboard or joystick based controls, as gestures provide much more degrees of freedom and give a more natural way of commanding the robot [1]. Voice commands are also proven to be a viable method to control the robot [2], but voice limits control of a robot to a specific language, which limits the user group to those, who know the language of a system. Gestures, on the other hand, require only a human body, which makes them more universal in a sense of possible users. That are the reasons why gesture detection systems are being developed. But gestures themself also can be used differently and can be described multiple ways.

3.1. Gestures

Gestures are expressive, meaningful body motions involving physical movements of the fingers, hands, arms, head, face, or body [3]. Gestures are generally categorized into three types:

1. hand and arm gestures: hand signals and sign languages
2. head and face gestures: emotions, direction of gaze, facial expression, head movement
3. body gestures: full body motion

Meaning behind a gesture can be described by the following factors [3]:

- spatial information: surroundings of gesture
- pathic information: how the gesture is made (the path it takes)
- symbolic information: sign, made by gesture
- affective information: emotions behind the gesture

Also gestures can be described by how many time frames they need to be described:

- Static gestures - gestures, which need only one time frame to be fully described, for example indicating a direction by pointing the arm in according direction.
- Dynamic gestures - gestures, which need a sequence of time frames to be fully described, for example waving a hand.

3.2. Robots controlled by gestures

Different robotic systems with availability of gesture control were created over the years. For example in [4] every part of the system, starting from the glove and finishing with the robot arm itself was developed completely from scratch with a goal to create the robot and control it wirelessly with a device, which is not the controller (Fig. 1). But the current trend is to use cameras and image processing to create gesture controlled systems, because almost any device has a camera with HD resolution or higher. One of such systems is presented in [5], where with the help of image processing a simulated robot was controlled in real time with hand gestures, such as can be seen in Fig. 2. Similar approach is used in [6], where hand gestures were used as a way to make UGV move to one of predetermined directions. There also exist much more complex solutions, with usage of machine learning algorithms and more complex sensors, than a camera, such as [7], where researchers used Microsoft Kinect V2 to build a robust system, which is the prototype to a safely controlled robot arm, which can work independently, but operator can take control any time and use hand gestures to manually control movement of a robot or to create a new task for it. (Fig. 3). Current systems tend to be proof of concepts, which shows that gestures are a viable way in human-robot communication and can be a more natural way of commanding a robot.

(a) (b)

Figure 1. *Example of gesture detection based application consisting of glove (a) and controllable by it robot arm (b)* [4]

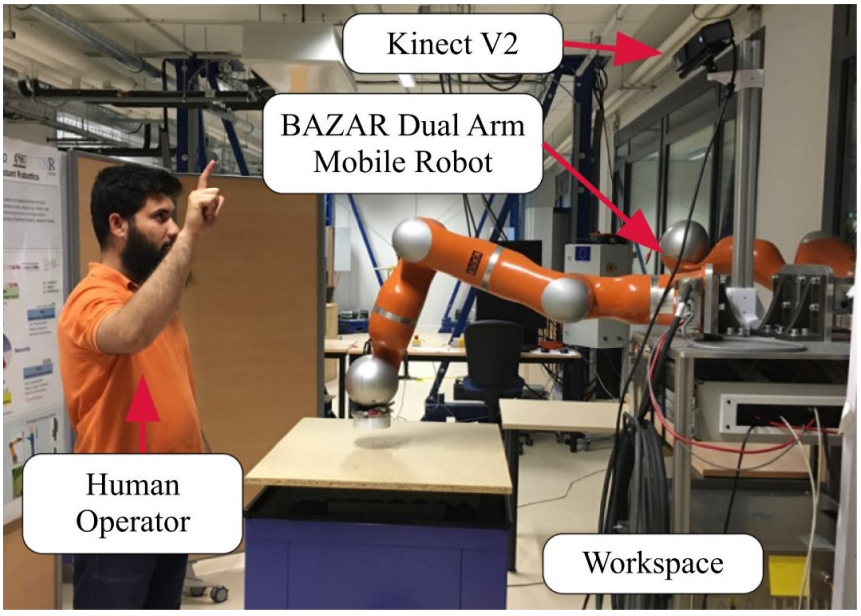


Figure 2. *Workspace set-up, used in* [7] *to interact with robot arm with gestures to make a safer environment for a human with control over some distance. Human operator stands in front of the Kinect V2 camera*

3.3. General design of gesture detection systems

Gesture processing in its core is an information processing task, which can be described by a four-stage model given in Fig 3 [8]. First, information is recorded by the sensor, the next step is to extract only information required for decision making, which is the next step. In the last step, this decision triggers certain response action.

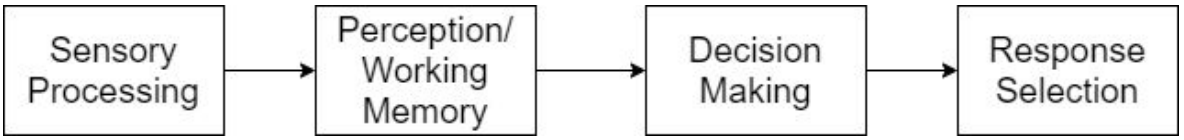


Figure 3. *Four-stage model of human information processing [8]. First, we receive information from the sensor. After that we pick and store necessary information. Next step is to make a decision, what this information represents and after that react accordingly.*

In [9] this model was extended to generally describe the process of gesture recognition for the Human-Robot Collaboration field in five steps (Fig 4):

1. Sensor Data Collection. Acquiring the information, received by the sensor or the system of multiple sensors. This step describes the choice of sensors for recording the information about the human.
2. Gesture Identification. Recognizing the gesture in the received data.
3. Gesture Tracking (Optional. In case detection of dynamic gestures is necessary). Track the location of the gesture to understand, if the gesture belongs to the same person.
4. Gesture Classification. From data, received on steps 2 and 3, classify gestures into known gestures.
5. Gesture mapping. Translating received results into command for the robot.

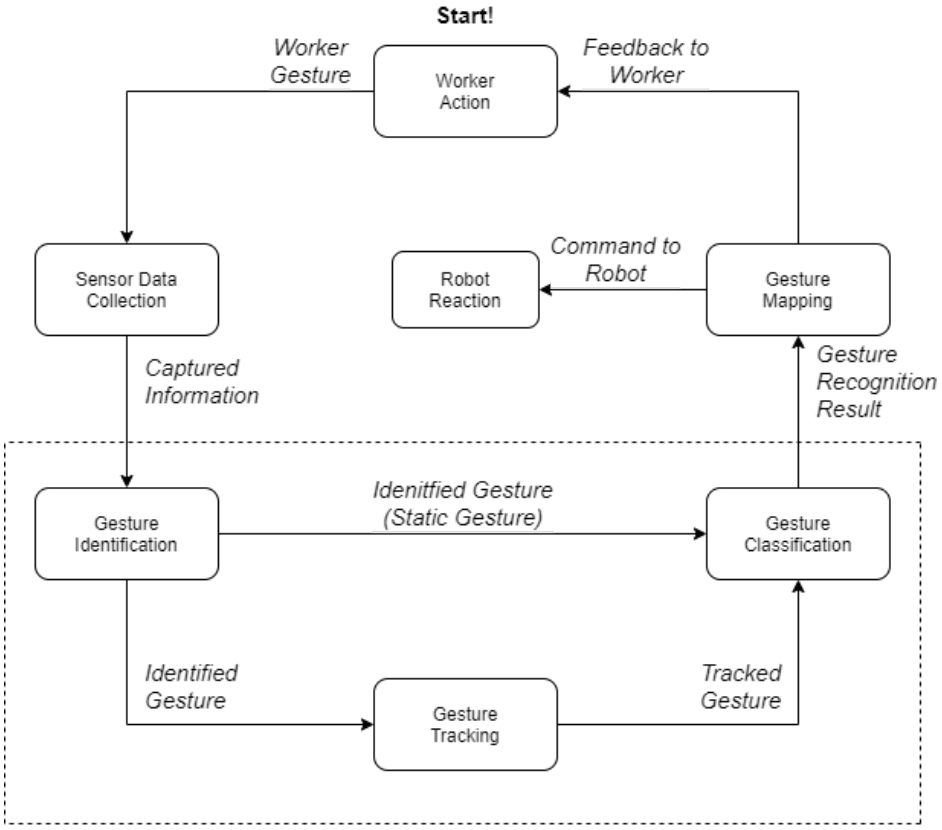


Figure 4. *Process model of gesture recognition for human-robot collaboration* [9]

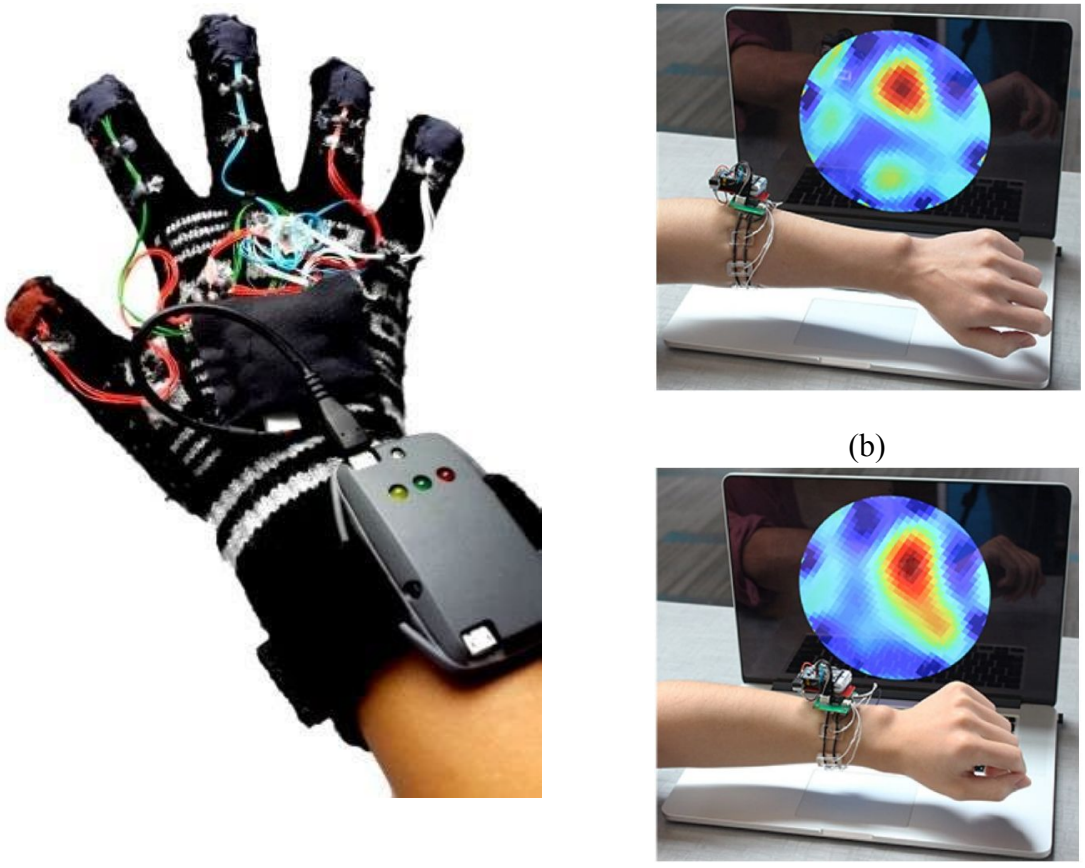
As each step of the gesture detection pipeline (Fig. 4) has several approaches on how they can be done, the following subsections will cover possible solutions for each step.

3.3.1. Sensors

The raw input data for detecting gestures is generally collected either from wearable or non-wearable sensors.

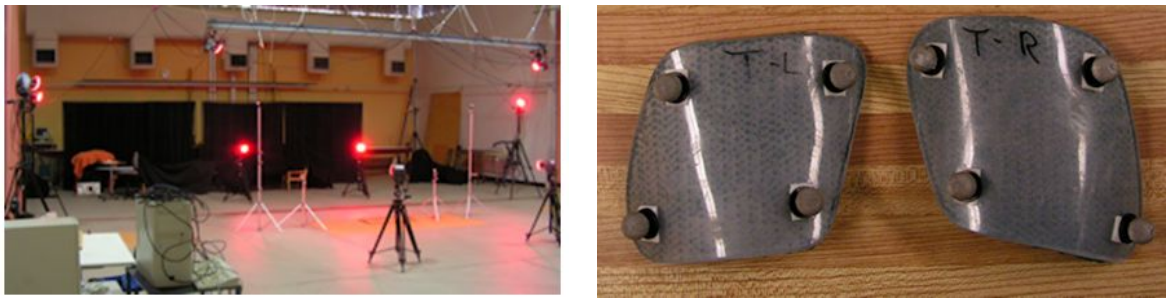
Wearable devices are glove-based (Fig. 5a), band-based (Fig. 5b and Fig. 5c) and marker solutions (Fig. 6a and Fig. 6b). Glove-based devices consist of a glove with multiple sensors, which can detect: position in space, orientation and flexing of fingers [1]. Band-based sensors are referred to as wristband solutions, which use tomography to measure the impedance of the hand in the wristband region. Each hand gesture has its own unique impedance values, which can be measured and analysed [10]. Marker system utilizes a combination of one or more cameras (Fig. 6a) and special markers (Fig. 6b), attached to the human body, which needs to be tracked. Depending on the type of the camera, output can be 2D RGB image with

certain color representing the tracked body parts [11], or it can be 3D RGB image, which gives point cloud image [12]. Marker based tracking systems are commonly used to create CGI characters with real mimics and movements (Fig. 7) [13].



(a) (b) (c)

Figure 5: (a) *Example of glove device from* [14]. (b) *Example of band (wristband) device from* [10] *worn on arm (b) and wrist (c) with corresponding reconstructed images of interior limb structure*.



(a) (b)

Figure 6. *Example of camera setup for motion capture with markers (a) and markers, attached to a plate, which stick to a human body (b)* [15].

(a) (b)

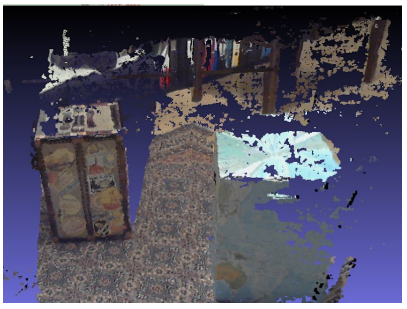
Figure 7. *Example of usage of Motion Capture in movies* [13]. *In (a) an actor in motion capture costume is acting as Gollum. With the help of motion capture costume CG Gollum (b) acting is made.*

Non-wearable devices can be based on 4 different sensor solutions: RGB camera (Fig 8a), stereo camera (Fig 8b), depth camera (Fig 8c) and radio frequency (RF) based sensors (Fig. 8d). The RGB camera's output is a regular 2D RGB (Fig 9a) image. Idea, which lies behind stereo cameras, is a way how people perceive depth. It is 2 cameras, taking each their own image (Fig 8b), which can be later used to construct a point cloud image. Depth sensors are

non-stereo depth sensing devices [9]. The output of a depth camera is always a depth image. In contrast with an ordinary RGB image, where each pixel value represents a color intensity, in depth image each pixel value represents a distance from a camera (Fig 8c). RF-based (radars) sensors are referred to the devices, which use radio frequency to detect objects [16]. Because certain RF can pass through objects, it is possible to use RF to track the person even through walls, as it was done in [16].

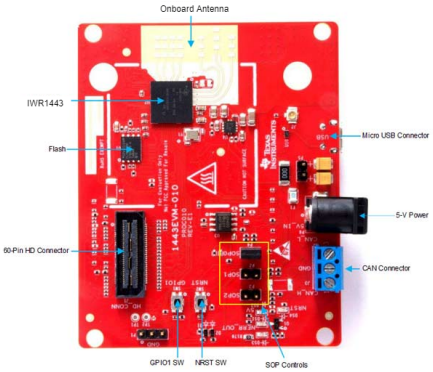
(a)

(b)



(c)

(d)



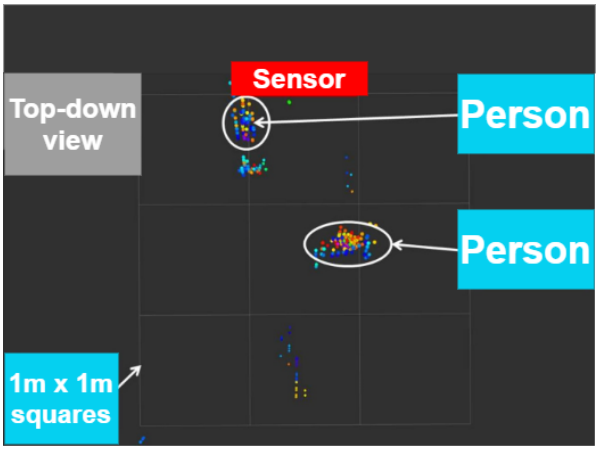


Figure 8. *Camera and output image from it (a); stereo camera and output in poincloud form* [17] *(b); depth camera and output depth images (darker - further) (c); Radar sensor and it's output in top-down view* [18] *(d).*

3.3.2. Gesture identification

The role of gesture identification task is to process the data captured by the sensors to extract relevant gesture information for the next gesture detection stage. Output of this step might differ, depending on the used type of sensor, but the current trend is to output keypoints which contain the information of different parts of the human body, such as shoulders, elbows and palms.

In glove-based and band-based sensors, the amount of data is defined by the amount of sensors connected to the glove, meaning that gesture identification step is already done during the creation of the glove sensor. On the other hand, non-wearable and marker based sensors provide data that requires extensive filterings as a person does not occupy the whole image and takes only a portion of it. There are several ways to extract the necessary information from this raw data, where each method works with any non-wearable sensor output, if not stated so. First method is identification by color, which is used with RGB images and can not be used with depth images as they don't represent colors or RF sensors output which the location of detected points. It can be used to extract the information about the limb, by knowing its color [19]. It is also used in combination with markers, as they are always of contrast color. [11].

Next method is model-based, where all necessary body parts are constructed into a body model. Later this model is fitted into the input image and information about found body parts becomes output [20], [21]. With emergence of depth sensors, this method also started using 3D models to make it more robust and efficient for detecting body parts [22]. Similar to that approach is usage of local features. Idea behind it is to divide the whole image into small regions without any correlation to body parts. After that some predetermined local features, such as high contrast or detected blob size, are found in every region and identification is made, based on the information about these local features [23]. Next way of extracting information is motion. In [24] such method was used to identify gestures on the image with static background. In [16] this solution was used in combination with an RF transmitter to detect and track the person through walls. Recent trend in gesture identification is to use

machine learning algorithms [9]. These algorithms at some point outputs keypoints of necessary joints [25]–[27].

OpenPose

One of the machine learning solutions is OpenPose. It is an open-source real-time system for multi-person 2D pose detection, including body, foot, hand and facial keypoints [28]. This software is free for non-commercial use [28], making it good identification software for the research, related to gesture recognition. OpenPose takes 2D RGB images as input and outputs detected keypoints location, associated with the joints of a person. OpenPose achieves multi-person real-time tracking by utilizing Part Affinity Field (PAF), which helps to combine all detected keypoints in multiple sets, where each set is a different person. PAF describes a limb by a set of vectors, which contain information about the location of limb joints (Fig. 10). OpenPose also supports multiple body models with different number of keypoints in each model (Fig. 9). The MPII (Max Planck Institute of Informatics) is one of the datasets with labeled human body models used for training machine learning algorithms for gesture identification. MPII model consists of 15 keypoints annotating ankles, knees, hips, shoulders, elbows, wrists, necks, torsos and head tops. COCO (Common Objects in Context) is a dataset of labeled images with human beings being one of labeled objects. Human body model was labeled for the COCO keypoint challenge, where the task is to create gesture identification software with output in a form of body keypoints. The COCO body model is similar to the MPII model, but also describes facial keypoints (eyes, ears and nose). Also OpenPose developers expanded the COCO body model with a more detailed foot model and called the modified version BODY_25.

Figure 9. *Three different body models, originating from different datasets* [28]. *MPII - Max Planck Institute for Informatics dataset. COCO - Common Objects in COntext* [29] *dataset.*

Figure 10. *Example how PAF looks. Vectors (orange) describe the location of left shoulder and left elbow [28].*

3.4.3. Gesture tracking

Gesture tracking is a process of tracking the gesture over multiple time frames and making sure that identified gesture belongs to the same person as in the previous frame. This step is required for dynamic gestures, as they require multiple time frames to be described. This step is unnecessary for gesture recognition with wearable sensors, as the sensor is worn by the

operator during the gesture detection cycle, meaning that the information from those sensors is always about the same person.

In [9] tracking is defined as the process of finding temporal correspondences between data frames, which is done by comparing the current frame with the prediction, made from previous frames. The prediction is generally achieved by utilizing algorithms such as mean shift, Kalman Filter (KF) and Particle filter (PF). Mean shift tries to find the same region of mean point (coordinate point from RF sensors or pixels from camera tipes) values from the previous frame in the new frame [30]. KF returns the predicted position by combining information from multiple time frames [31], to which the current frame can be compared. PF is able to keep track of multiple identified people at the same time considering different features on the image as particles and giving them a weight, thus showing how important this particle is for description of the tracked object. With this it becomes possible to find the same person in multiple pictures by finding the same particles. [32].

3.4.4. Gesture classification

Gesture classification is a task where identified gestures are given a specific meaning, e.g., an identified upright hand motion is classified as a stop signal. Gesture classification is generally achieved with the help of machine learning algorithms, such as K-Nearest Neighbors (KNN), Hidden Markov Model (HMM), Support Vector Machine (SVM), Dynamic Time Warping (DTW) and Artificial Neural Networks (ANN).

KNN finds the closest match between the provided example data and the received one [33].

HMM is an unobservable Markov chain with input states (X) and transition probabilities between them (a), which gives output probabilities (b) and output observations (O) (Fig.11) [34]. Because HMM keeps track of transitions between states, it is used for dynamic gesture recognition [35].

SVM (Fig. 12) consists of data, which should be segregated and hyperplane, which divides the space into two regions, representing the desired groups and segregates the data into these two groups [37].

DTW is an algorithm that aligns the input and example sequences without depending on time [38]. DTW has been extensively applied for voice recognition, but it can be generally used to compare any sequences of data. In [39] this algorithm was used on all body keypoints (model is similar to MPII on Fig. 7, with 2 more keypoints for feet) with giving weight for all keypoints sequences depending on the gesture, so that DTW results would take only necessary keypoints into account to classify each gesture.

ANNs consist of one or multiple layers of connected artificial neurons (Fig. 13a). Input layer brings the data into the network, the hidden layer performs calculations in neurons of that layer, and the output layer is the last layer, where output from each neuron is observed. Each neuron is a function with multiple inputs and outputs. In (Fig. 13b) example neuron will take values of x1 and x2 as inputs and perform mathematical operation which is summation in this case and output the result to the next NN layer [40]. Connection between neurons and functions they contain is determined during the training phase.

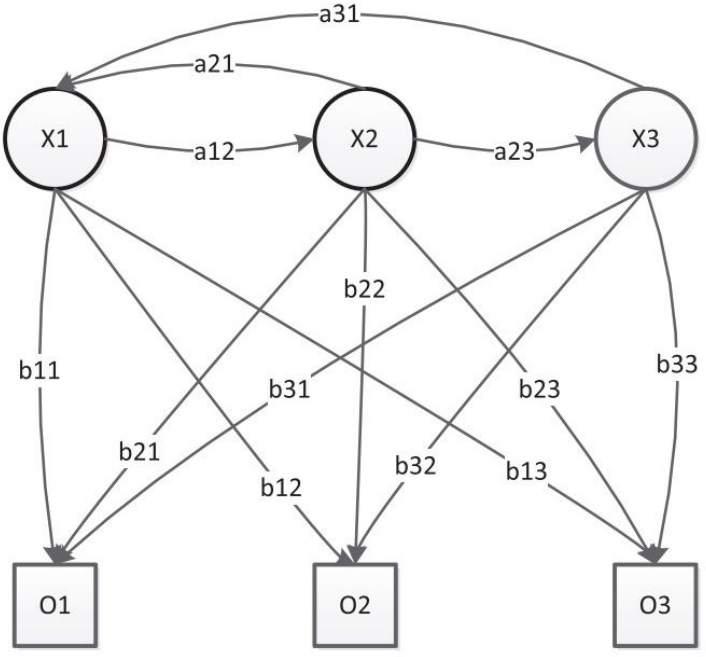


Figure 11. *Example of Hidden Markov Model* [41]

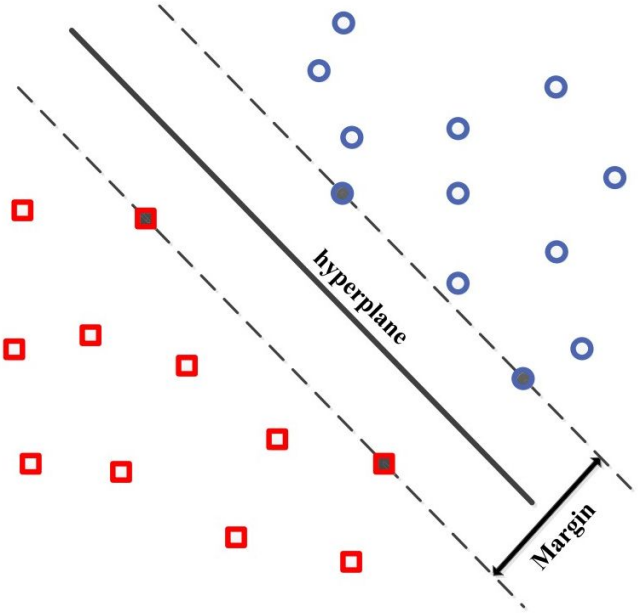
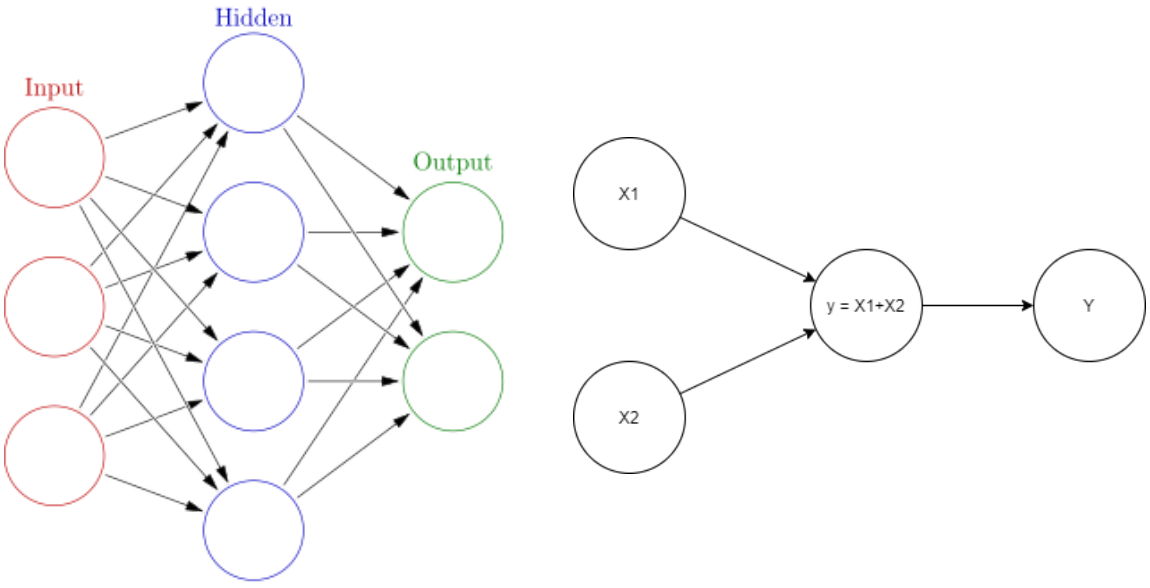


Figure 12. *Example of Support vector machine, consisting of hyperplane, which segregates objects into two groups and segregated objects* [37]



(a) (b)

Figure 13. *Illustrated structure of an Artificial Neural Network (a) and Example of Artificial Neuron (b)* [40]

3.4.5. Gesture mapping

Last step of the whole process is to send the classified gesture to the robot and translate it into a set of commands.

Classified gesture is sent to the robot control script, where the next action is executed based on the received gesture. Gestures can be used to manually control a robotic arm [4] or UGV [6], or to trigger a set of actions [5].

It is desirable to reuse developed gesture detection systems for different robots, meaning that it should be written with a universal and highly used robotic system in mind.

For that purpose there exists a Robot Operating System (ROS). ROS is a data distribution software framework, which allows multiple computers to exchange information between each other as messages with predefined variable types [42]. Figure 14 shows how messages are described in ROS as an example of message type "Point", which describes the position of point in space. Every variable in message type is described by giving variable type and then the name of variable.

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

Figure 14. *ROS message definition for geomtry_msgs/Point*

ROS allows programs to communicate with each other even when running on different computers and being written in different programming languages. The fact that ROS multi-lingual allows many already developed non ROS programs to be adapted to ROS, commonly referred to as wrapping. . For example, Table 1 presents the wrappers for OpenPose.

Wrapper	Quality of documentation	Latest commit
firephinx[1]	Covers installation of wrapper and on what versions of software and drivers it was tested. Has short description on how to work with it and solutions for common problems	29.04.2019
stevenjj[2]	Covers which version of OpenPose was tested. Has installation manual and explains how to work with it	29.11.2017
solbach[3]	Covers installation of wrapper	07.06.2017
ims-robotics[4]	no available documentation	29.05.2018
ravijo[5]	Has links to ROS wrappers for supported cameras and OpenPose. Doesn't have operating systems and hardware list, with which the wrapper would work.	10.10.2019

Table 1. *ROS-wrappers for OpenPose compared by documentation coverage and last commit to the project (information checked at 17.10.2019).*

[1] https://github.com/firephinx/openpose_ros
[2] https://github.com/stevenjj/openpose_ros
[3] https://github.com/solbach/openpose-ros-tue
[4] https://github.com/ut-ims-robotics/openpose-ros
[5] https://github.com/ravijo/ros_openpose

4. REQUIREMENTS

4.1 Objective

The objective of this thesis is to create a gesture classification software, which integrates OpenPose for gesture-based control of robots.

4.2 System requirements

4.2.1 Functional requirements

1) A camera is used for gesture detection
2) Gesture classification that is able to detect at least 5 gestures
3) New gestures that need to be detected can be defined without changing the source code
4) Ability to detect static gestures
5) The gestures can be used to send all the basic steering commands (e.g., move forward, steer left, steer right, turn left, turn right, move backwards, and stop) to a mobile robot.
6) The gesture detection should not take more than 200 ms
7) Should be able to run on middle grade consumer PC (intel i5 processor, nvidia GTX 1050 graphics card) or higher

4.2.2 Non-functional requirements

1) OpenPose ver 1.5.1 or higher
2) ROS Kinetic or newer
3) Ubuntu 16.04 or newer

4.2.3 Gesture requirements

Three sets of gestures were made. First set of gestures is based on aircraft marshalling [43] (table 2). Second set of gestures is based on cycling signals [44] (table 3). Third set of gestures was made by the author as part of the thesis (table 4).

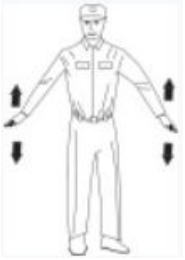
Command	Original command picture	Command as static gesture
Forward		
Left		
Right		
Stop		
Slow down (or Backward)		

Table 2. *Set of gestures, based on aircraft marshalling signals. First column describes the meaning of gesture (with a possible alternative for robot control). Second column shows the original gestures. Third column shows versions modified for static gesture recognition.*

Command	Original command picture	Command as static gesture
Right		
Right (Alternative)		
Left		
Left (alternative)		
Stop (Backward)		

Table 3. *Set of gestures, based on cyclist hand signals. First column describes the meaning of gesture (with a possible alternative for robot control), Second column shows the original gestures. Third column shows real life examples.*

Command	Command as static gesture
Left	
Right	
Stop	
Forward	
Backward	

Table 5. *Set of gestures, made from combining gestures from two previous sets into a custom control set. First column describes the meaning of gesture. Second column shows how the following gesture should look like.*

5. INTEGRATION AND BENCHMARKING OF OPENPOSE IN ROS

The fundamental requirement for this thesis is the integration of OpenPose in a ROS-based system for controlling a robot. In order to assess the capabilities and limitations of OpenPose for such a task, the quality of available ROS wrappers was evaluated and the performance of OpenPose was benchmarked on different computational systems commonly deployed in robotics.

5.1 Quality of ROS wrappers for OpenPose

When OpenPose was initially released, several wrappers were made, but during the lifetime of OpenPose, its API was changing and some wrappers didn't work with the newer versions of OpenPose. This means that it is essential for the wrapper to be as recent as possible to be sure that it works. It is also preferable, that wrapper has good documentation and installation instructions, so it is possible to set up for a user, who has no prior experience with ROS. Wrapper should also work with 2D RGB image ROS topics.

Wrapper	Issues	Advantages
firephinx	Models other then BODY_25 doesn't work: user needs to find a way to publish images through ROS topics	Updated; Has a good manual; All values that can be changed for OpenPose can be changed in the wrapper. Works on recent versions of OpenPose.
stevenjj	was not tested on any versions other than 1.0.0 (no hands detection); no support for BODY_25	Was not tested, so unknown
solbach	not understandable how to install through manual; was not tested on any versions other than 1.0.0 (no hands detection); no support for BODY_25	Was not tested, so unknown
ims-robotics	repository didn't download properly	Was not tested, so unknown
ravijo	Support versions 1.5.0 and higher. Right now only works with intel realsense cameras	most recent wrapper. Doesn't consume a lot of computing power for 3d body detection.

Table 6. *Quality of existing ROS wrappers for OpenPose*

(information checked at 17.10.2019)

5.2 Benchmarking OpenPose

OpenPose was tested on several computer specifications to see how the configuration of the system affects the gesture recognition performance (Table 7). First column describes the system, the second one describes configuration of OpenPose. Different body models and net resolutions, to which the input image is downscaled, were tested. Net resolution is a resolution of an image, which is processed inside a neural network. The higher the net resolution, the higher the accuracy of body keypoints estimation can be expected, as the processed image is more detailed. -1 in net resolution means that ratio is configured automatically by openpose to best fit the aspect ratio of input image (144x-1 will mean 144x80 pixels if input image aspect ratio is 16x9). In the third column are output frame rate results for different OpenPose configurations on different computer specifications.

Tested PC	Condition (BODY_25 - 25 keypoints, COCO - 17 keypoints; Stock - net resolution 656x368, BODY_25)	Results
i7 8700, 16GB RAM, GTX1070Ti	GPU mode stock	15-16 FPS
	GPU mode stock with face and hand	5-6 FPS
	CPU_ONLY 656x368 mode only body BODY_25	0.1 FPS
i5 7300HQ 8GB RAM	CPU_ONLY mode net_resolution 128x96 COCO	6.3 FPS accuracy is low, (random objects are detected as human)
	CPU_ONLY mode net_resolution 128x96 BODY_25	1.1 FPS
	CPU_ONLY mode net_resolution -1x256 COCO	1 FPS
NVidia Jetson TX2	GPU mode stock	1.5 FPS
	net resolution 128x96 COCO model	8 FPS
	net resolution 128x96 BODY_25	12.5 FPS
	net resolution 144x-1 BODY_25	9.3 FPS
	net resolution -1x256 BODY_25	3 FPS
Nvidia Jetson Nano	GPU mode stock	Computer froze
	net resolution 128x96 BODY_25	7.0 FPS accuracy is low (random objects detected as human)
	net resolution 144x-1 BODY_25	5.8 FPS
	net resolution -1x256 BODY_25	Computer froze
NUC	net_resolution -1x128 COCO	2.2 FPS

Table 7. *Results of benchmarking OpenPose on different computers and different CNN net_resolutions*

6. DESIGN

The gesture based robot control system, proposed for this work, is depicted in Fig. 14. Camera records the person making a gesture in front of it and a ROS wrapper for usb connected cameras (usb_cam) publishes the captured images on a ROS topic “usb_cam/image_raw”. The OpenPose ROS wrapper, having subscribed to “usb_cam/image_raw” topic, receives the images, calculates the keypoints and publishes them on “/openpose_ros/human_list” topic. After that classification software (classifier) subscribes to the topic with keypoints to take them and obtains user defined reference gestures from a YAML file. These keypoints are used to classify the gesture into one of predetermined commands from a YAML file. After that classified gesture is published to "openpose_ros/found_pose" topic, to which robot control script is subscribed to it and take the command and make the robot move accordingly.

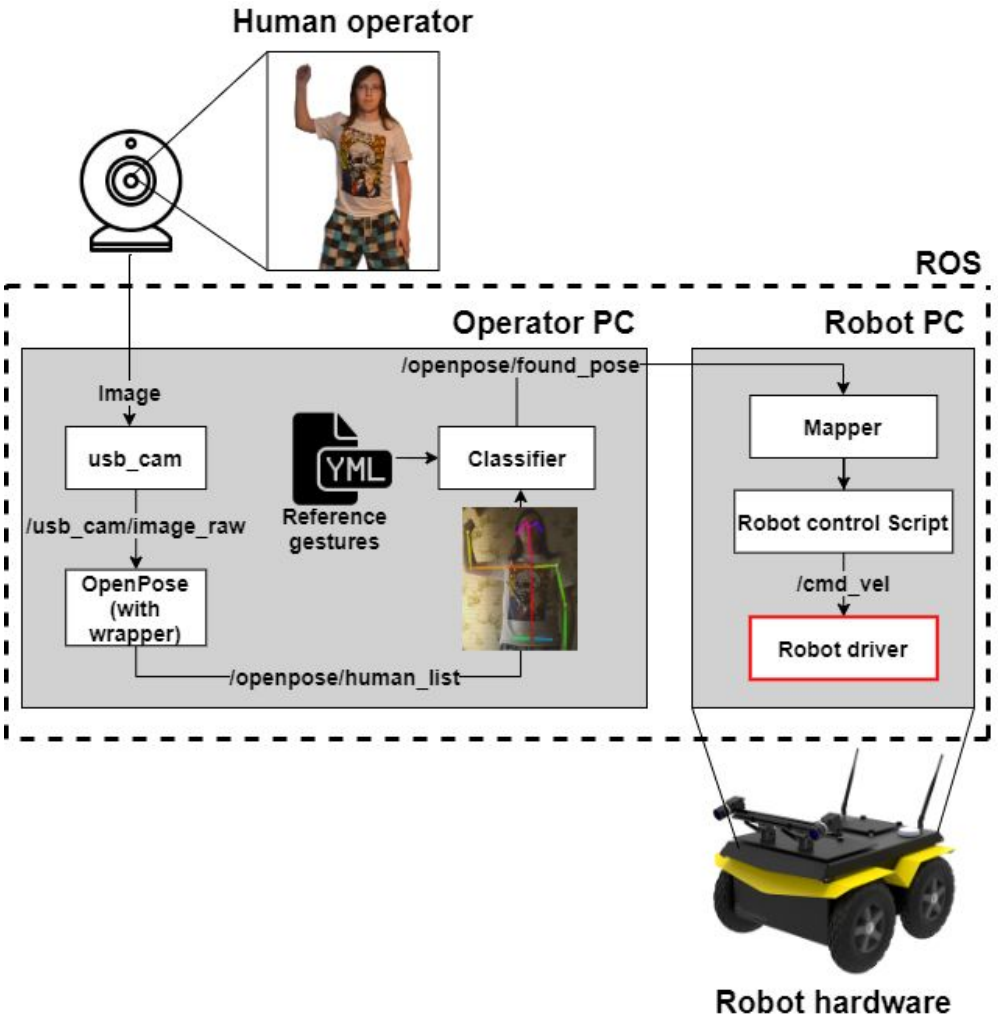


Figure 14. *Overview of proposed gesture based robot control system*

6.1 Camera

ROS package usb_cam requires Video4Linux (v4l),which is a collection of drivers and API for cameras to work with linux. Any camera, which is supported by v4l will work in the system.

6.2 OpenPose

Firephinx OpenPose ROS wrapper was chosen for this work, as at the moment of creating the system it was the only wrapper, which was working with the up-to-date version of OpenPose and supported 2D images. This wrapper subscribes to the image topic and uses received images as input for OpenPose is able to to receive images. Wrapper also receives the output

from OpenPose and creates the topic, to which it publishes the keypoints of all detected people in the image in the format, shown in Figure 15. OpenPoseHumanList message type consists of four message types. OpenPoseHumanList message describes the number of humans on each input image frame and after that each human is described by the next message type. Order of description is by how long the detected person was on the frame, meaning that the person who was detected for the longest consecutive time will be the first to be described in OpenPoseHuman. Described features are:

- number of detected keypoints of body/face/hands
- location of body and face on the image in the form of bounding box
- location of all keypoints on the picture

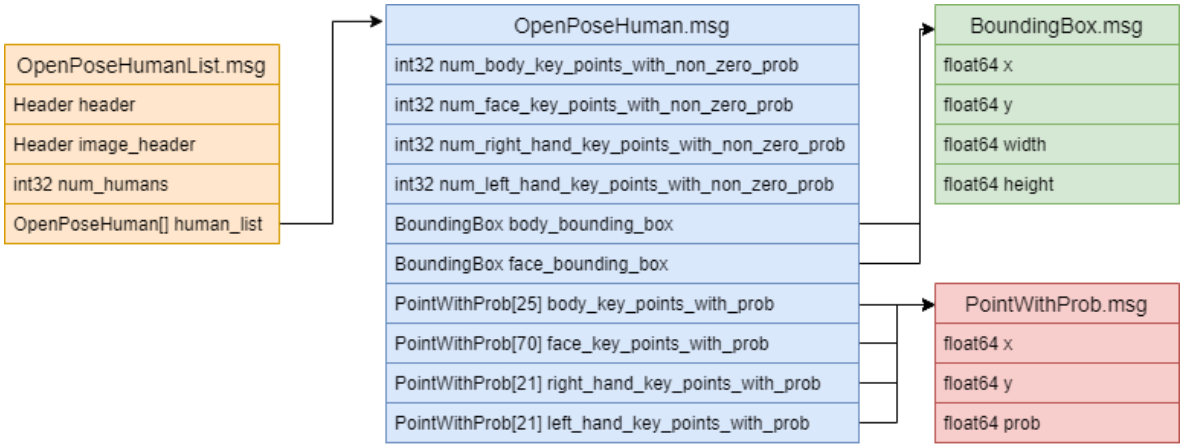


Figure 15. *Structure of message sent by OpenPose ROS wrapper*

Body keypoints are stored in the set, consisting of 25 pixel coordinates with detection probability value. Coordinates are ordered the same way as the OpenPose provided keypoints , seen in Figure 16.

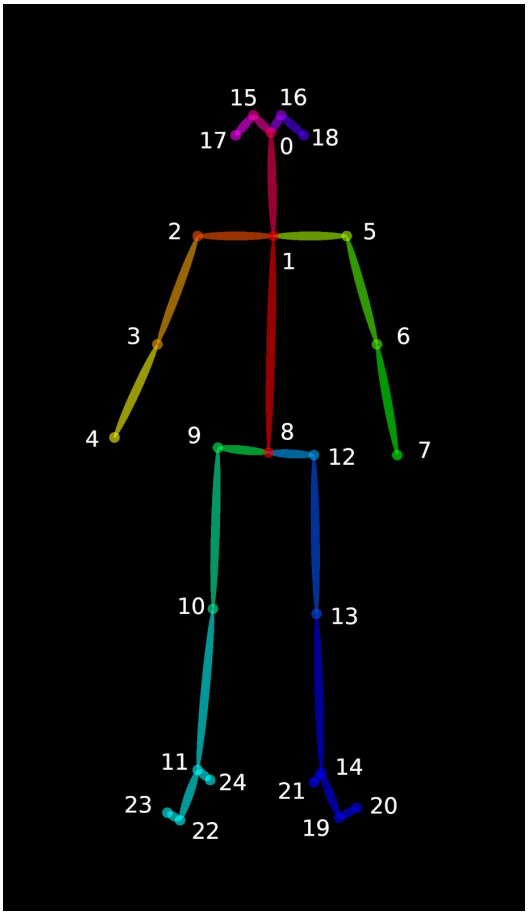


Figure 16. *The order of OpenPose keypoints [45]*

6.3 Classifier

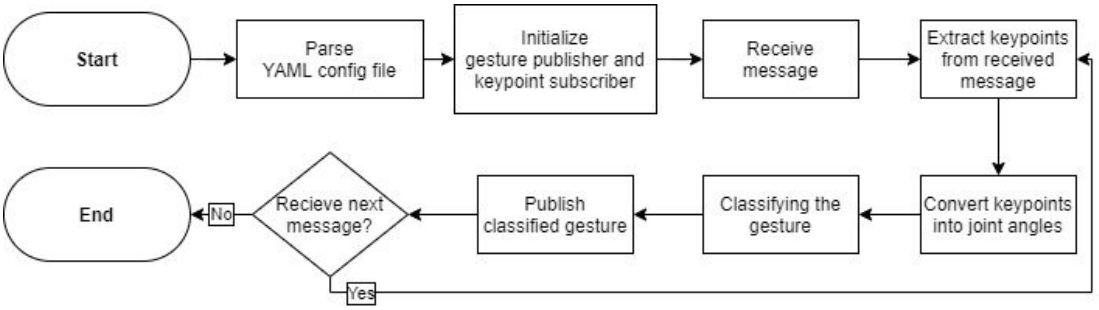


Figure 17. *Design of the gesture classifier.*

Classifier takes the detected keypoints and outputs the classified arm gesture. Figure 17 shows the workflow how it is done. When the classifier is launched, it parses the YAML file with reference gestures (Fig. 19) to obtain the reference gestures information. After that ROS

gesture publisher and keypoint subscriber are initialized and keypoints from the received message are extracted. After that the extracted keypoints are converted into joint angles. These joint angles and parsed information are used to classify the gesture. Result of this process is published on the ROS topic.6.3.1 Normalization with joint angles

As keypoints from OpenPose are pixel coordinates, it means that size of the person and coordinates will vary depending on the position of the human in the frame. For that reason this information should be normalised. As the normalization method classifier uses joint angles, which do depend on relative position of keypoints.

Keypoints to joint angles

To classify arm gestures, four joints are require and to describe them 7 keypoints are needed:

1 - Neck

2 - Right Shoulder

3 - Right Elbow

4 - Right Wrist

5 - Left Shoulder

6 - Left Elbow

7 - Left Wrist

Figure 18 shows how the aforementioned 7 keypoints are transformed into joint angles. First step is to create a pair of vectors, whose axis origin is the keypoint, which describes the location of the joint itself. In case of arm gestures, this will be the following pairs:

1.1 - Right Shoulder-Neck

1.2 - Right Shoulder-Right Elbow

2.1 - Right Elbow-Right Shoulder

2.2 - Right Elbow-Right Wrist

3.1 - Left Shoulder-Neck

3.2 - Left Shoulder - Left Elbow

4.1 - Left Elbow - Left Shoulder

4.2 - Left Elbow - Left Wrist

By using the following pairs of vectors the joint angle is calculated as shown in equation 1.

$$\alpha = atan(\frac{v_{1x}*v_{2y} - v_{1y}*v_{2x}}{v_{1x}*v_{2x} + v_{1y}*v_{2y}}) \quad (1)$$

Where v_1 describes the x and y coordinates of the first vector in pair and v_2 describes the x and y coordinates of the second vector in pair.

By doing this operation on all the pairs, we get 4 angles:

1 - right_shoulder

2 - right_elbow

3 - left_shoulder

4 - left_elbow

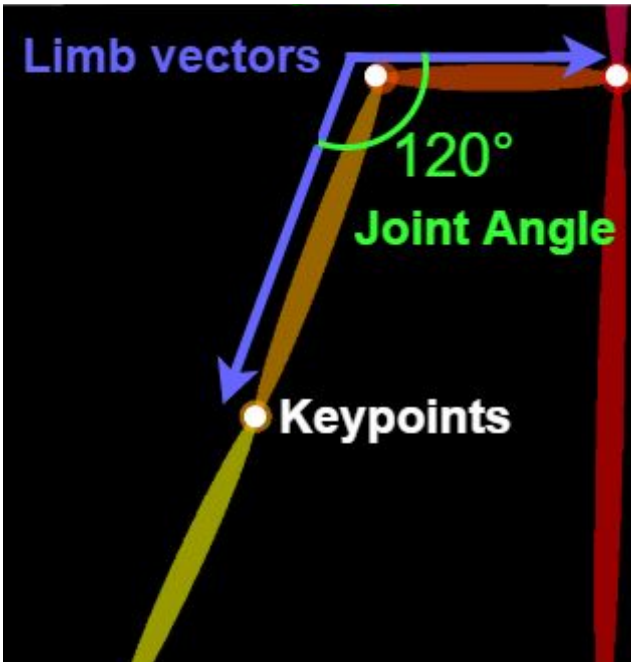


Figure 18. *Workflow of calculating the angle. Visualization of calculating the joint angles.*

6.3.2 Reference Gestures YAML Notation

For the classifier to classify the gesture, file with reference gestures and their names is provided. File is written in YAML markup language and has the following format of description (Fig 19):

Gestures show the YAML parser, where to look for gestures. **Gesture_name** describes the name for the following gesture and what will be returned, if this gesture is detected.

Angle_value_degress - float value, which describes the angle, which joint should have. Can be any float value in between 0 - 360 degrees.

Error_value_dergees - float value, describing, how inaccurate can the operator make the gesture. The bigger the value, the bigger inaccuracy is tolerated. To ignore the joint, anything bigger than 180 should be placed.

```
1   gestures:
2   - gesture_name: "gesture_name"
3     right_shoulder:
4       angle: angle_value_dergees
5       error: error_value_dergees
6     right_elbow:
7       angle: angle_value_dergees
8       error: error_value_dergees
9     left_shoulder:
10      angle: angle_value_dergees
11      error: error_value_dergees
12    left_elbow:
13      angle: angle_value_dergees
14      error: error_value_dergees
```

Figure 19. *Structure of the reference gestures YAML file*

6.3.3 Classifying the gesture

Together with reference angles, measured angles are taken into the KNN-inspired part of the classifier.

Idea behind the algorithm is to compare measured joint angles with reference angles of each gesture and choose the one with highest similarity. Nearest neighbor in this case is reference angles and K is equal to one.

The whole process is consisting of one nested loop (Fig. 20). Before the loop is created, 3 variables are generated. First one is float with value 0 (`S1`), second one is float with value bigger than 720 (`S2`) and the third one is string with value "nothing" or any other text, which classifier should output if there is no gesture (`detected_gest`).

(`S1`) will store the mean error value for the currently checked gesture, (`S2`) will store the calculated mean error value for the currently chosen gesture and (`detected_gest`) will store the name of this gesture.

First loop is started and s1 will reset to 0 every loop iteration. We take the first set of reference angles (`reference_angles`) and error values (`error_values`) from the and start the second loop, where we find how different are the reference angle value and measured angle value. Variable (`ignored angles`) is created to keep track of how many joint angles were ignored during the second loop. During every iteration of the first loop is reset to 0.

Second loop starts and the first elements from (`angle_values`), (`reference_angles`) and (`error_values`) are taken:(`m_angle`), (`ref_angle`), (`ref_error`).

First, (`ref_error`) value is checked. If it is bigger than 180, (`ignored angles`) is increased by one and the next iteration of the second loop is started immediately.

Difference between reference angle and measured angle is calculated (`error`). To exclude the problem, where angle 0 and 359.9 are too far away in calculation, but in reality they are close, we check if the calculated difference is bigger than 180. If yes, then we subtract the

calculated difference from 360 and get the real difference between angles. If not, we already have the real difference.

After that (`error`) is compared to (`ref_error`). If (`error`) is bigger, the next iteration of the first loop is started immediately. Else (`error`) is added to (`S1`) and the next iteration of the second loop is started.

After the second loop is finished, (S1) is divided by the number of angles, which were checked (`checked_angles`) to get an average error (`average_error`). This (`average_error`) value is compared to (S2) value. If (`average_error`) is smaller, (S2) will be overwritten by (`average_error`) and (`detected_gest`) will be overwritten by the name of gesture, which was checked in the current loop iteration.

Next iteration of the loop is started and the whole process continues for the next reference gesture in the list.

When the first loop is finished, (`detected_gest`) is sent to the publisher.

```
gestures_name = set of gesture names
gesture_values = nested set of angle values for each gesture
gesture_errors = nested set of error values for each angle in each gesture
angle_values = set of calculated angle values
S1 = 0.0
S2 = 180.0
detected_gest = “nothing”

for i in size(gesture name)
{
   S1 = 0.0
   reference_angles = set number i from gesture_values
   error_values = set number i from gesture_errors
   ignored angles = 0
   for j in size(ref_angles)
   {
      m_angle = element number j from angle_values
      ref_angle = element number j from reference_angles
      ref_error = element number j from error_values
      if ref_error > 180
      {
         ignored angles = ignored angles + 1
         go to next iteration of second for loop
      }
      error = absolute(m_angle - ref_angle)
      if error > 180
      {
         error = 360 - error
      }
      if error > ref_error
      {
         S1 = 0.0
         go to next iteration of first for loop
      }
      S1 = S1 + error
   }
   checked_angles = size(ref_angles)-ignored angles
   average_error = s1/checked_angles
   if average_error < s2
   {
      S2 = average_error
      detected_gest = element number i from gestures_name
   }
}
return detected_gest
```

Figure 20. *Design of KNN-based part of gesture classifier*

7. RESULTS

7.1 System setup

All tests were done on the system with i5 7300HQ laptop CPU, 8GB of RAM and GTX 1050 with 2GB of video memory. Camera resolution was 1280x720 and frame rate was 30 FPS.

Operating system was Ubuntu 16.04 with ROS Kinetic and OpenPose version 1.5.1.

Because of the amount of video memory, net resolution for OpenPose was lowered to 224x128 to be able to run.

The implemented classifier code with gesture sets can be found on GitHub[6]

7.2 Test results for proposed system:

OpenPose framerate - 10 FPS

New gesture was detected every 100 ms seconds.

Classifier work time - 256 - 324 nanoseconds.

Tables 8-10 represent the output of OpenPose and gesture classifier for three proposed gesture sets.

[6] https://github.com/ut-ims-robotics/openpose_gesture_control

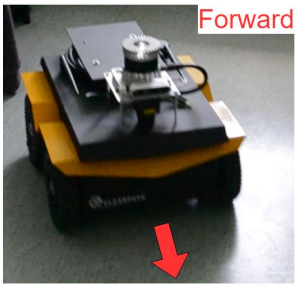
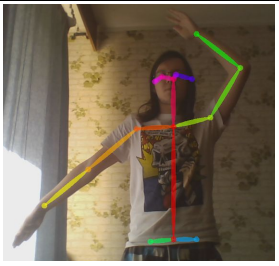
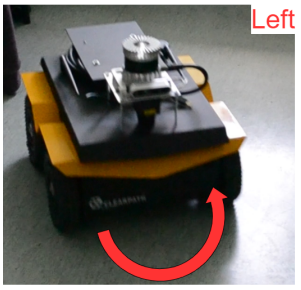
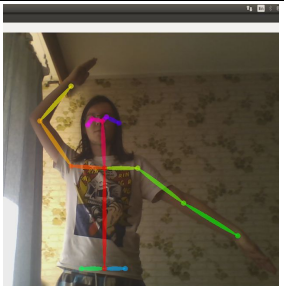
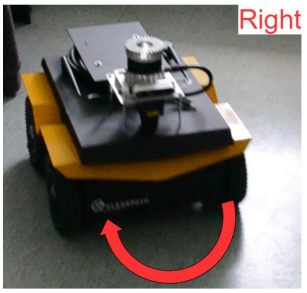
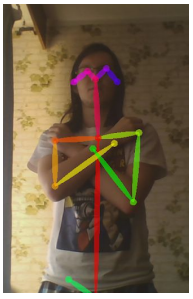
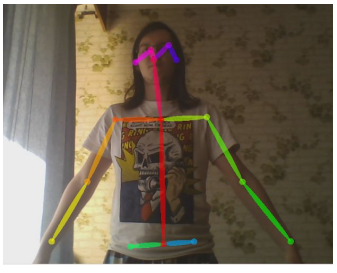
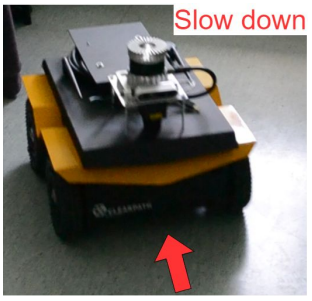
Name of the gesture	Output From OpenPose	Output from classifier
Forward		
Left		
Right		
Stop		
Slow down (or Backward)		

Table 8. *Result of gesture detection for the first set of gestures with output from OpenPose, classifier and movement from robot*

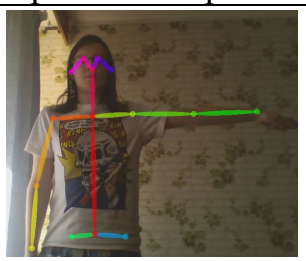
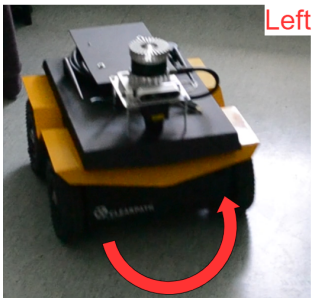
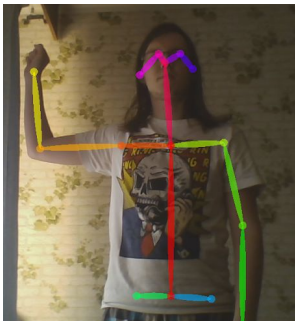
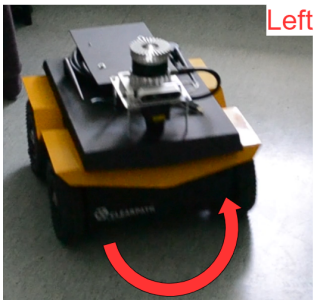
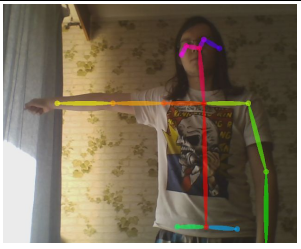
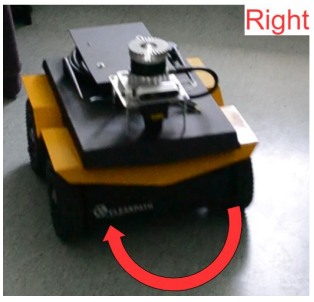
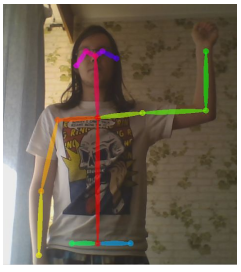
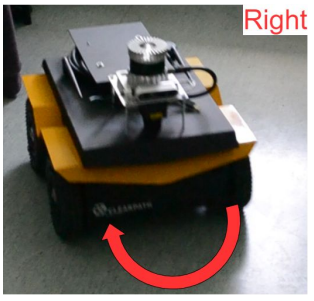
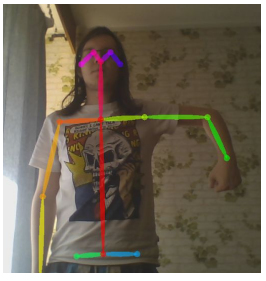
Name of the gesture	Output From OpenPose	Output from classifier
Left		
Left (alternative)		
Right		
Right (alternative)		
Stop (Backward)		

Table 9. *Result of gesture detection for the first set of gestures with output from OpenPose, classifier and movement from robot*

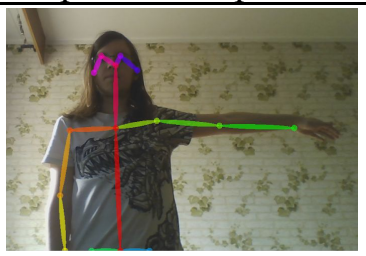
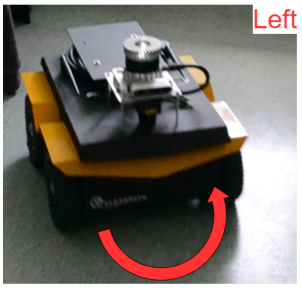
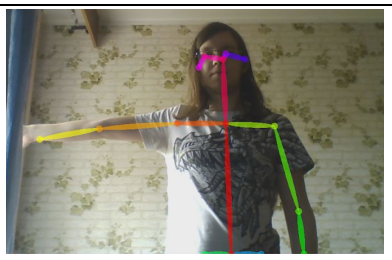
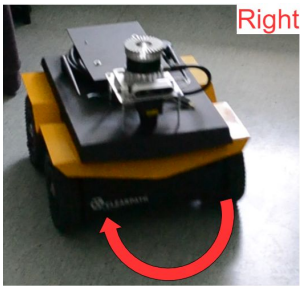
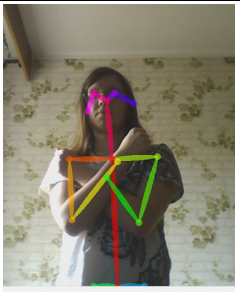
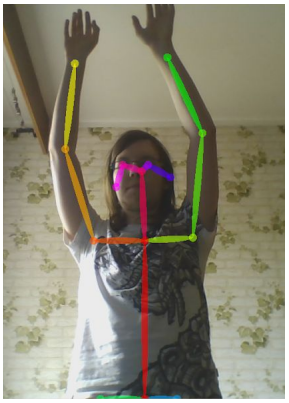
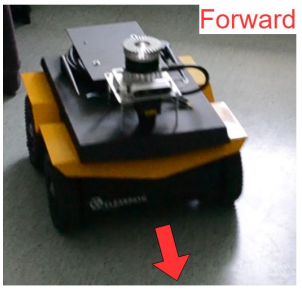
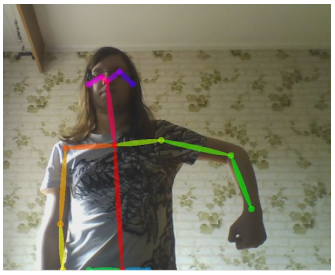
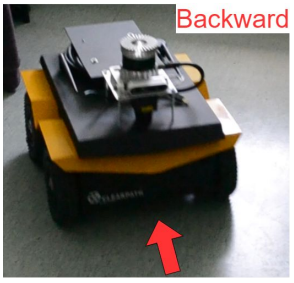
Name of the gesture	Output From OpenPose	Output from classifier
Left		
Right		
Stop		
Forward		
Backward		

Table 10. *Result of gesture detection for the first set of gestures with output from OpenPose, classifier and movement from robot*

7.3 Discussion

The implemented gesture classification system was able to classify all required gestures, but OpenPose had problems with detection of limb intersections, as can be seen on the test of the last gesture set. This problem becomes even bigger if the intersection is not in front of the camera. That is the reason that the stop signal from air marshalling was changed to be done not above the head, but on the chest. Possible solutions for this problem is to move the camera higher. Air Marshals show those signals to the pilots, who are always located higher than him. Second option is to run OpenPose on a more powerful system, which would be able to handle higher net resolutions, so the intersection would be visible for OpenPose. But this situation also showed that the proposed normalization method still provides enough information for gestures to be detected. We can see that as long as the direction of the limb is detected correctly, angles will be calculated also correctly and the result will be correct.

Main bottleneck of the tested system was hardware, which was not able to run OpenPose. Because of this combination OpenPose was able to have only 10 FPS, which meant that new keypoints were published only every 100 milliseconds. Classifier by itself didn't use any complex algorithms and required only up to 324 nanoseconds, but classified gesture was published only every 100 milliseconds, which is the publish rate of OpenPose. The best option to increase speed and not lose in accuracy is to use more powerful hardware for openpose to run, but eventually the first limiting factor will be the framerate of the camera, and the next limiting factor will be the bandwidth of how fast ROS messages can be published.

Current set of gestures is limited by arms. Proving that joint angles can actually be used to describe gestures, the next step is apply this description method on hand gestures. Currently there are two obstacles for implementation of hand gesture detection. First one is hardware requirements for finger detection in OpenPose. As an example, test system, which was used in this thesis, is not able to run OpenPose with hand gesture identification because there is not enough video memory. Second problem is that it is significantly harder to find orientation of the hand without a depth image.

Second problem is that the classifier cannot detect dynamic gestures. KNN can not be used alone for dynamic gestures, as it works only with one time frame, and another algorithm should be used. With current normalization method angle values can be recorded in the sequence. Possibly good options here will be DTW and HMM. DTW is used for sequence matching for years and stored sequence changing joint angles values can be used to determine if the following movement is not random and has a meaning. HMM can become suitable because it can keep track of state transitions, which allows HMM to be built on top of already existing KNN and to use it as an input to keep track of dynamic gestures.

8. SUMMARY

Outcome of the thesis is constructed gesture detection system, which uses an RGB camera in combination with OpenPose and allows the user to steer the UGV robot with predetermined gestures. Gestures in the system are described by angle values of joints, which are needed to make the gesture. Predetermined gestures can be changed without the need to change source code. Gesture detection system can control any robot, which uses ROS for communication and control.

9. BIBLIOGRAPHY

[1] T. G. Zimmerman, J. Lanier, C. Blanchard, S. Bryson, and Y. Harvill, "A hand gesture interface device," p. 4, 1987.

[2] Xiaoling Lv, Minglu Zhang, and Hui Li, "Robot control based on voice command," in *2008 IEEE International Conference on Automation and Logistics*, Qingdao, China, Sep. 2008, pp. 2490–2494, doi: 10.1109/ICAL.2008.4636587.

[3] S. Mitra and T. Acharya, "Gesture Recognition: A Survey," *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 37, no. 3, pp. 311–324, May 2007, doi: 10.1109/TSMCC.2007.893280.

[4] S. Verma, "Hand Gestures Remote Controlled Robotic Arm," p. 6.

[5] J. L. Raheja, R. Shyam, U. Kumar, and P. B. Prasad, "Real-Time Robotic Hand Control Using Hand Gestures," in *2010 Second International Conference on Machine Learning and Computing*, Feb. 2010, pp. 12–16, doi: 10.1109/ICMLC.2010.12.

[6] H. Kumar, V. Honrao, S. Patil, and P. Shetty, "Gesture Controlled Robot using Image Processing," *Int. J. Adv. Res. Artif. Intell.*, vol. 2, no. 5, 2013, doi: 10.14569/IJARAI.2013.020511.

[7] O. Mazhar, B. Navarro, S. Ramdani, R. Passama, and A. Cherubini, "A real-time human-robot interaction framework with robust background invariant hand gesture detection," *Robot. Comput.-Integr. Manuf.*, vol. 60, pp. 34–48, Dec. 2019, doi: 10.1016/j.rcim.2019.05.008.

[8] R. Parasuraman, T. B. Sheridan, and C. D. Wickens, "A model for types and levels of human interaction with automation," *IEEE Trans. Syst. Man Cybern. - Part Syst. Hum.*, vol. 30, no. 3, pp. 286–297, May 2000, doi: 10.1109/3468.844354.

[9] H. Liu and L. Wang, "Gesture recognition for human-robot collaboration: A review," *Int. J. Ind. Ergon.*, vol. 68, pp. 355–367, Nov. 2018, doi: 10.1016/j.ergon.2017.02.004.

[10] Y. Zhang and C. Harrison, "Tomo: Wearable, Low-Cost Electrical Impedance Tomography for Hand Gesture Recognition," in *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15*, Daegu, Kyungpook, Republic of Korea, 2015, pp. 167–173, doi: 10.1145/2807442.2807480.

[11] A. Bellarbi, S. Benbelkacem, N. Zenati-Henda, and M. Belhocine, "Hand gesture interaction using color-based method for tabletop interfaces," in *2011 IEEE 7th International Symposium on Intelligent Signal Processing*, Floriana, Malta, Sep. 2011, pp. 1–6, doi: 10.1109/WISP.2011.6051717.

[12] N. S. Pollard, J. K. Hodgins, M. J. Riley, and C. G. Atkeson, "Adapting human motion for the control of a humanoid robot," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, May 2002, vol. 2, pp. 1390–1397 vol.2, doi: 10.1109/ROBOT.2002.1014737.

[13] L. Karreman, "The Motion Capture Imaginary: Digital renderings of dance knowledge," 2017.

[14] M. A. Ahmed, B. B. Zaidan, A. A. Zaidan, M. M. Salih, and M. M. bin Lakulu, "A Review on Systems-Based Sensory Gloves for Sign Language Recognition State of the Art between 2007 and 2017," *Sensors*, vol. 18, no. 7, p. 2208, Jul. 2018, doi: 10.3390/s18072208.

[15] D. Robertson, "Vicon Motion Capture." Jul. 18, 2013.

[16] F. Adib and D. Katabi, "See through walls with WiFi!," p. 12.

[17] "Producing 3D point clouds with a stereo camera in OpenCV," *Stackable*, Apr. 27,

2014. https://erget.wordpress.com/2014/04/27/producing-3d-point-clouds-with-a-stereo-camera-in-opencv/ (accessed May 17, 2020).
[18] "mmWave sensors in robotics: enabling robots to 'sense & avoid.'" Accessed: May 10, 2020. [Online]. Available: https://training.ti.com/sites/default/files/docs/mmwave_in_robotics_part1_1.pdf.
[19] J. Letessier and F. Bérard, "Visual tracking of bare fingers for interactive surfaces," in *Proceedings of the 17th annual ACM symposium on User interface software and technology - UIST '04*, Santa Fe, NM, USA, 2004, p. 119, doi: 10.1145/1029632.1029652.
[20] H. Hamer, K. Schindler, E. Koller-Meier, and L. Van Gool, "Tracking a hand manipulating an object," in *2009 IEEE 12th International Conference on Computer Vision*, Kyoto, Sep. 2009, pp. 1475–1482, doi: 10.1109/ICCV.2009.5459282.
[21] I. Oikonomidis, N. Kyriazis, and A. A. Argyros, "Markerless and Efficient 26-DOF Hand Pose Recovery," in *Computer Vision – ACCV 2010*, vol. 6494, R. Kimmel, R. Klette, and A. Sugimoto, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 744–757.
[22] I. Oikonomidis, N. Kyriazis, and A. Argyros, "Efficient model-based 3D tracking of hand articulations using Kinect," in *Procedings of the British Machine Vision Conference 2011*, Dundee, 2011, pp. 101.1-101.11, doi: 10.5244/C.25.101.
[23] D. Weinland, R. Ronfard, and E. Boyer, "A survey of vision-based methods for action representation, segmentation and recognition," *Comput. Vis. Image Underst.*, vol. 115, no. 2, pp. 224–241, Feb. 2011, doi: 10.1016/j.cviu.2010.10.002.
[24] R. Cutler and M. Turk, "View-based interpretation of real-time optical flow for gesture recognition," in *Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition*, Nara, Japan, 1998, pp. 416–421, doi: 10.1109/AFGR.1998.670984.
[25] F. Zhang, X. Zhu, H. Dai, M. Ye, and C. Zhu, "Distribution-Aware Coordinate Representation for Human Pose Estimation," *ArXiv191006278 Cs*, Oct. 2019, Accessed: Feb. 18, 2020. [Online]. Available: http://arxiv.org/abs/1910.06278.
[26] M. Kocabas, S. Karagoz, and E. Akbas, "MultiPoseNet: Fast Multi-Person Pose Estimation Using Pose Residual Network," in *Computer Vision – ECCV 2018*, vol. 11215, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, pp. 437–453.
[27] J. Zhang, Z. Chen, and D. Tao, "Human Keypoint Detection by Progressive Context Refinement," *ArXiv191012223 Cs Eess*, Oct. 2019, Accessed: Apr. 28, 2020. [Online]. Available: http://arxiv.org/abs/1910.12223.
[28] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields," *ArXiv181208008 Cs*, May 2019, Accessed: Feb. 18, 2020. [Online]. Available: http://arxiv.org/abs/1812.08008.
[29] T.-Y. Lin *et al.*, "Microsoft COCO: Common Objects in Context," *ArXiv14050312 Cs*, Feb. 2015, Accessed: Apr. 30, 2020. [Online]. Available: http://arxiv.org/abs/1405.0312.
[30] D. Comaniciu, V. Ramesh, and P. Meer, "Real-time tracking of non-rigid objects using mean shift," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*, Hilton Head Island, SC, USA, 2000, vol. 2, pp. 142–149, doi: 10.1109/CVPR.2000.854761.
[31] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *J. Basic*

Eng., vol. 82, no. 1, pp. 35–45, Mar. 1960, doi: 10.1115/1.3662552.
[32] K. Okuma, A. Taleghani, N. de Freitas, J. J. Little, and D. G. Lowe, "A Boosted Particle Filter: Multitarget Detection and Tracking," in *Computer Vision - ECCV 2004*, vol. 3021, T. Pajdla and J. Matas, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 28–39.
[33] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, p. 1883, 2009.
[34] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989, doi: 10.1109/5.18626.
[35] J. McCormick, K. Vincs, D. Creighton, S. Hutchison, and S. Nahavandi, "Teaching a Digital Performing Agent: Artificial Neural Network and Hidden Markov Model for recognising and performing dance movement," p. 6.
[36] S.-Z. Yu, "Hidden semi-Markov models," *Artif. Intell.*, vol. 174, no. 2, pp. 215–243, Feb. 2010, doi: 10.1016/j.artint.2009.11.011.
[37] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995, doi: 10.1007/BF00994018.
[38] M. Müller, "Information Retrieval for Music and Motion. Chapter 4: Dynamic Time Warpind," 2007.
[39] "Gesture Recognition using Skeleton Data with Weighted Dynamic Time Warping:," in *Proceedings of the International Conference on Computer Vision Theory and Applications*, Barcelona, Spain, 2013, pp. 620–625, doi: 10.5220/0004217606200625.
[40] S. S. Haykin and S. S. Haykin, *Neural networks and learning machines. Introduction*, 3rd ed. New York: Prentice Hall, 2009.
[41] M. Elmezain, A. Al-Hamadi, J. Appenrodt, and B. Michaelis, "A Hidden Markov Model-based continuous gesture recognition system for hand motion trajectory," in *2008 19th International Conference on Pattern Recognition*, Tampa, FL, USA, Dec. 2008, pp. 1–4, doi: 10.1109/ICPR.2008.4761080.
[42] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," p. 6.
[43] "Aircraft marshalling," *Wikipedia*. Nov. 09, 2019, Accessed: Apr. 30, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Aircraft_marshalling&oldid=925358314.
[44] "CAA National." https://www.caa.ca/bike/on-the-road-cyclists/riding-skills-tips/ (accessed May 19, 2020).
[45] "OpenPose github webpage," *GitHub*. https://github.com/CMU-Perceptual-Computing-Lab/openpose (accessed Apr. 29, 2020).

NON-EXCLUSIVE LICENCE TO REPRODUCE THESIS AND MAKE THESIS PUBLIC

I, Igor Rybalskii,

(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Gesture Detection Software for Human-Robot Collaboration,

(title of thesis)

supervised by Robert Valner and Karl Kruusamäe.

(supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

author's name Igor Rybalskii

20/05/2020