JACOBS
UNIVERSITY

Human-Machine Interaction Through Gesture Recognition (on the example of divers' sign language)

by

Dmitrii Cucleschin

Bachelor Thesis in Computer Science

Prof. Andreas Birk
Name and title of the supervisor

Date of Submission: March 5, 2015

Jacobs University — School of Engineering and Science

With my signature, I certify that this thesis has been written by me using only the indicates resources and materials. Where I have presented data and results, the data and results are complete, genuine, and have been obtained by me unless otherwise acknowledged; where my results derive from computer programs, these computer programs have been written by me unless otherwise acknowledged. I further confirm that this thesis has not been submitted, either in part or as a whole, for any other academic degree at this or another institution.

Signature Place, Date

Abstract.

Nowadays, we live in the world, where machines get more and more advanced and are tailored for a variety of different tasks. To initiate one of such tasks, person has to communicate the desired function to a machine through some sort of input device. Unfortunately, in some scenarios, common input devices (like keyboard and mouse) are unavailable and one has to implement a different communication method.

In this thesis, I will be focusing on developing and testing one of such methods - communication with the machine via hand gestures. There are number of cases, where such method is more convenient than the analogues, but here we will focus on the specific case of integrating such a system in an underwater robot. Robot will be equipped with stereo camera and will be taught to recognize basic diver sign language. That will allow him to monitor health and safety of the human diver, as well as to receive instructions regarding the tasks and further steps it needs to perform from a distance.

Contents

1 Introduction.

1.1 Motivation.

Robotics department of Jacobs University Bremen has been working on several projects involving underwater robots. One of such projects, CADDY (Cognitive Autonomous Diving Buddy), is meant as an assistant for the diver, allowing to transport objects, take photographs and scan the surrounding area. However, equipment like this can be quite bulky and can possibly put the diver in danger in critical situations. Therefore, a concise communication method has to be implemented between human and robot to initiate tasks and report current danger status. Since CADDY is equipped with a depth camera, in this project we will be focusing on implementing one of such methods - communication using hand gestures. Equipped with an ability to recognize diver's sign language, CADDY will be able to be controlled from a safe distance, as well as to help diver and notify the others in case of any emergencies.

1.2 Research Question.

Based on the problem discussed above, the following research question arises, that I will be solving thorough my bachelor's thesis project:
Can a robust gesture-based communication system be implemented for a use in the underwater robot?
Since in a system like this reliability is very critical, it isn't enough to just implement the application logic. Thorough testing has to be performed to make sure all of the components of the system are functioning as expected. Therefore, the scope of this project isn't limited to research and implementation, but also quality assurance and creating proper design specifications.

1.3 Expected Research Contribution.

As a result of this research project, CADDY can be equipped with gesture recognition software, which will definitely be a very valuable addition to already impressive features of the device. With easy extensibility, it will be possible to extend the recognized gesture set and reuse this software in different projects. I feel that enabling people to communicate better and more naturally with the machine may simplify a lot of routine tasks, as well as potentially improve the speed of such communications, which may be crucial in the event of emergency situations.

1.4 Outline.

This proposal briefly describes the steps that were taken to implement a working diver's sign language recognition using a depth sensor (on an example of Kinect v2.0). The introduction presents the problem, as well as potential benefits of choosing this particular method. Next, state of the art section describes the overall progress and development in gesture recognition area, featuring some notable examples and experiments. Organizational issues, such as the project timeline, code license, deliverables and evaluation

criteria are discussed. Final application, as well as bachelor's thesis will be developed according to specifications defined in this research proposal paper.

2 State of the art.

Robot development has been advancing a lot over the course of last 10 years. We transcended all the way from simple programmable robots to truly autonomous machines, that can perceive the environment around them and interact with it. A lot of research has been done to perfect the communication with these "new-generation" robots. In 2006, a report [1] has been published, that shows that natural interaction (speech recognition, face recognition, gesture processing, etc.) is a great way of performing such a task, because of variety of input data and emotional feedback as a result of such a communication. Furthermore, artificial intelligence researchers [2] have discovered that when gesture recognition is present, humans tend to engage more with a robot and have better reviews, regarding their experience. However, such natural interfaces are not only useful for generic communication, but in some cases also for therapy. Robot AURORA [3] aims to aid children with autism by being a salient observer with changeable behavior. Researchers note, that the emotional feedback is one of the most important variables in this scenario and natural interaction through speech and gestures help to achieve that.

However, beyond research projects, the idea of implementing interface for consumer devices, controlled by natural interactions, isn't novel. At the moment, more and more manufacturers are investigating the possibility of adding voice or gesture control in their devices or applications. When the first Microsoft Kinect came out for Xbox 360, people were skeptical about the applications of a depth camera. Skeletal recognition and hand tracking were still quite choppy (partially due to hardware specifications and partially due to beta software) and controls using the camera were not intuitive and fluid. However, with the release of Xbox One and updated Kinect sensor, gesture control became one of the easiest ways to interact with the console. Kinect recognizes one of the three basic gestures: open hand, closed hand (fist) and lasso (2 fingers open), and bases interactions with the console on these. IR sensor with a better resolution allows for recognition in low- to no- light conditions. However, Microsoft isn't the only company, that is interested in providing such features. In their newest lineup of Smart TVs, Samsung introduced a feature [4], which allows basic control of your television with the gestures. Using a regular camera, that is built-in in the unit, TV can recognize simple gestures, like flip movement, thumbs up, selection etc.

However, the potential of this technology isn't limited to just entertainment. Number of startups are working on the devices, which would allow developers to embed gesture control in any application (most notable examples are Leap Motion sensor and Myo armband). With the powerful SDKs, it's just a matter of time until the technology is perfected enough for gestural interaction to become an everyday part of our lives. There are already some impressive experiments using the power of gestures: Microsoft Research has presented a Kinect sign language translator for American and Chinese languages [5], as well as a tool for the hospitals that allows touchless interaction with the computer during surgeries [6].

As we can see, even though the technology is still maturing and is not available to a

lot of people, number of useful applications based on it is growing every day and the overall potential is very high.

3 Preparation.

3.1 Overview of Kinect 2.0.

For the implementation of this project we will need a depth camera, that is able to produce a clear depth frame for the following hand segmentation and processing. While there are number of options to choose from, I chose **Microsoft Kinect 2.0**, because of its widespread availability and impressive specifications.

Second version of Kinect improved a lot over its predecessor, offering the following features:

- Color frames at 15-30 fps, depending on the light conditions (1920x1080)
- Depth frames at 30 fps (512 x 424) with improved fidelity
- Infrared frames at 30 fps (512 x 424)
- Skeletal tracking of up to 6 users (25 joints per person)
- Sound capture with an array of microphones

Kinect uses the following coordinate systems, that will be applied thorough the course of implementation of the project:

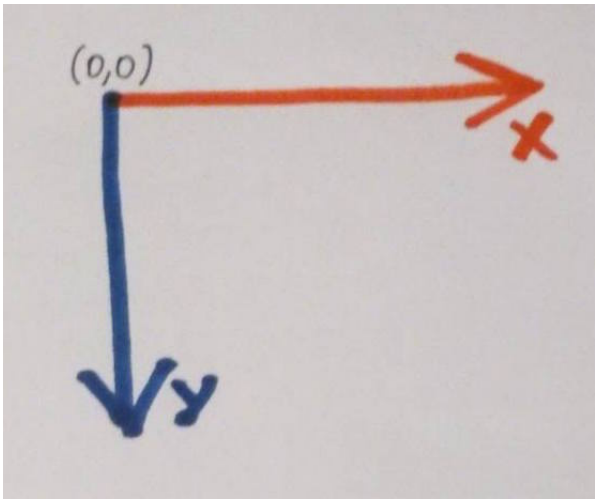

Figure 1: Coordinate system for a 2D depth frame

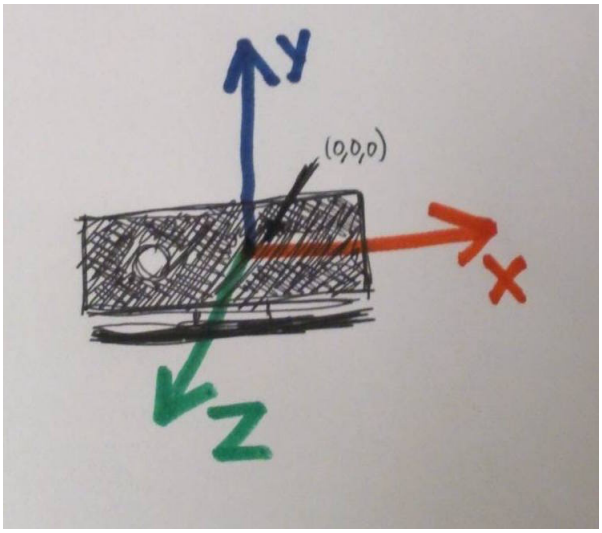

Figure 2: Coordinate system for a 3D skeleton tracking frame

3.2 Choosing SDK.

Microsoft has developed an **in-house SDK**, that is available as free download for every registered Windows/Kinect developer. It is used to take full advantage of all hardware capabilities of the sensor and includes a lot of helper utility functions, that ease the processing of the data. Drawback to using this approach is strict software limitation, allowing to only use it on Windows 8+ in the native C# environment (wrappers for Java and other languages exist, but are not mature enough in their implementation to suffice for this project).

Alternative option is using **libfreenect2**, a UNIX library based on libusb, that allows basic interfacing with Kinect. The current version allows the export of raw unprocessed color, depth and IR data. It also requires basic reprogramming of Kinect sensor to bypass the software requirements and has no methods for matching frames to each other (converting points from one coordinate space to the other). Typically, as done with the first version of Kinect sensor, this data is later used with a generic processing library like **OpenNI** to calculate skeleton data and allow the use of other features. However, Kinect v2 is still on very early stages of support in OpenNI, not allowing the full skeletal tracking, needed for the scope of this application. Experiments also proved, that frame rate is far below fluent (average 3-5 fps), which is not suitable for this project.

As a result, we will base the implementation on native SDK, to allow us to unlock the full potential of the Kinect hardware, while allowing the fluid UI. We will also include **EmguCV**, a C# wrapper of OpenCV to allow us to use otherwise complex algorithms, like Contours Detection, Convex Hull and Minimum Area Rectangle.

3.3 Gesture set

The following basic gesture set has been selected for the purposes of this project. Default semantics of those gestures are described, and the basic actions are assigned to them, based on some of the suggestions presented in CADDIAN language draft [7]. Let's separate all the gestures into two kinds: **static**, corresponding to a single hand expression and **dynamic**, building upon the static ones, but also tracking hand animation (such as movement or a sequence of changing states). This list is meant only as the initial draft and should be easily extensible, once more features are needed.

3.3.1 Static gestures

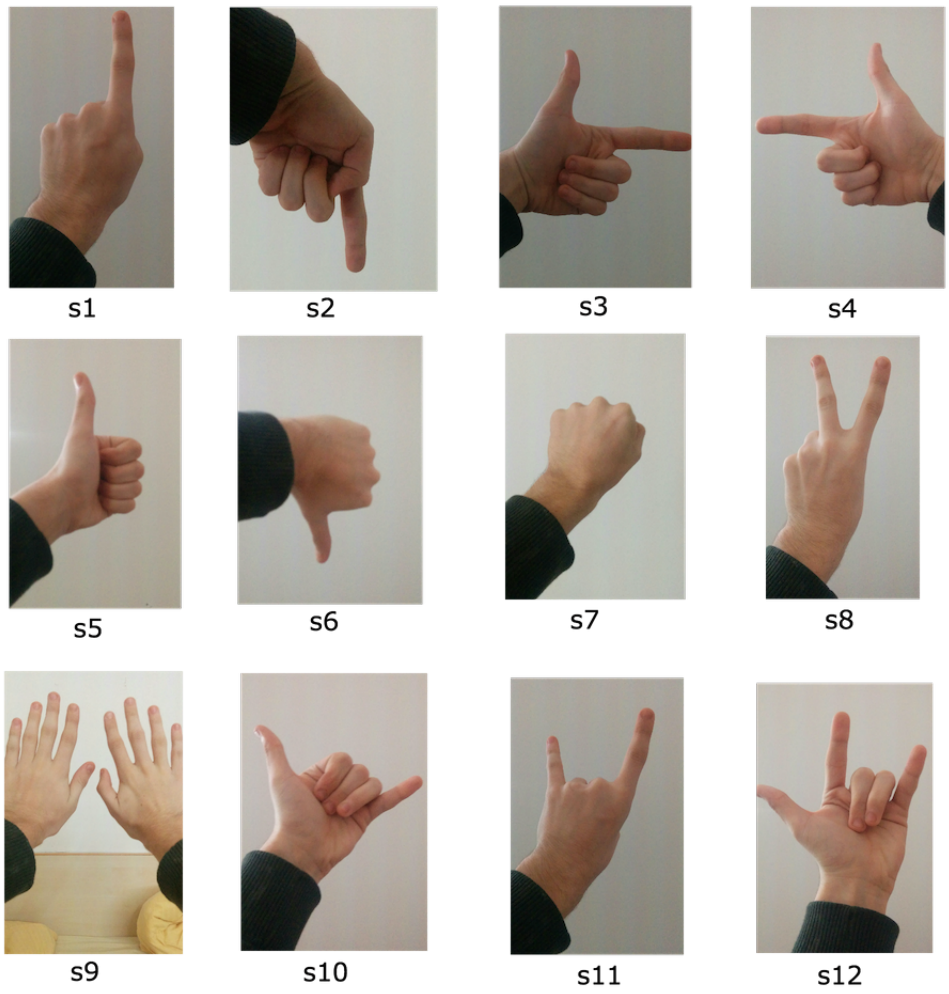

Figure 3: Static gesture set

- **s1.** Ascend. The longer user keeps the gesture, bigger the priority.
- **s2.** Descend. The longer user keeps the gesture, bigger the priority.

- **s3.** Turn $90°$ to the right.
- **s4.** Turn $90°$ to the left.
- **s5.** Confirm action / Accelerate.
- **s6.** Cancel action / Decelerate.
- **s7.** Switch between piloting and task management (for gestures s6 and s7).
- **s8.** Predator (for example, shark) has been spotted, use caution.
- **s9.** Abort the current operation.
- **s10.** Contact the crew with operation logs and message from the diver.
- **s11.** Take a picture.
- **s12.** Take a picture and a 3D scan of surrounding area.

3.3.2 Dynamic gestures

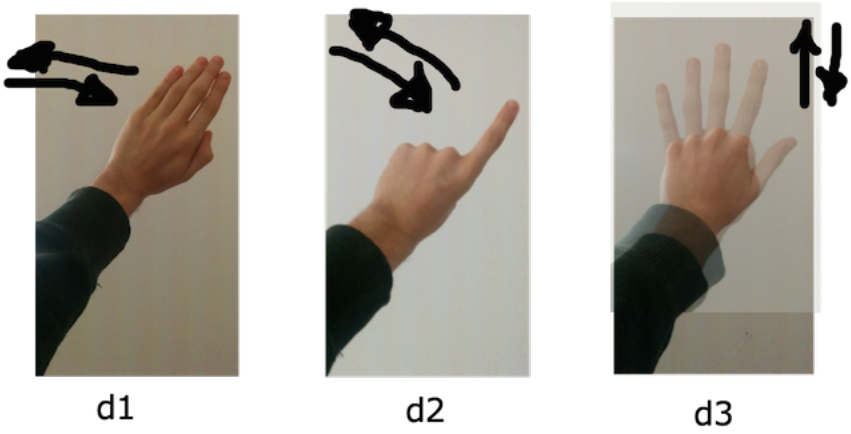

Figure 4: Dynamic gesture set

- **d1.** Emergency. Immediate abort of operations and ascend are requested. Faster the movement, bigger the priority.
- **d2.** Warning. Pause the operation and analyze the surroundings, contact the crew. Faster the movement, bigger the priority.
- **d3.** Closing and opening fist rapidly. Indicates that the oxygen is running out.

3.4 Environment assumptions.

For the purpose of this research project, the following assumptions will be made about the setup and the environment:

- Kinect is position to minimize the background noise (uncertain values).
- User is located directly in front of the sensor.
- View is not obstructed by any object in front of the user.
- There is only one user in the frame (multiple user tracking can be easily added, but adds computational intensity).
- When performing a gesture, hand is extended at least 20cm in front of the body.
- Gesture recognition will be less robust in the scenario of a fast moving hand, due to the motion blur produced by the sensor

3.5 System requirements.

As a result of choosing our depth camera and the SDK, the following system requirements need to be enforced to ensure the correct functionality of the application.

- 64-bit dual-core processor
- Dedicated USB 3.0 controller (Kinect will use its full capacity)
- $>$ 4 GB of RAM
- Graphics card with support of DirectX 11
- Windows 8 or newer
- Kinect v2 SDK and EmguCV installed

3.6 Code License

All the code, written as the part of this research project will be available under **GPL v2** license. After the final submission, code will be available freely as an open-source repository hosted on Github. Anyone is free to use, modify or distribute this software and its parts, as long as they provide an attribution back and license it under the same terms.

4 Implementation.

4.1 Project structure.

- **MainWindow**
  Includes rendering of the main application window, initializing Kinect sensor and displaying the results of the recognition.
- **Utility**
  Static class, including most global constants and helper functions, needed in the other classes.

- **HandRecognizer**
  Being initialized with an overall depth frame, this class is responsible for returning the segmented hand's binary mask.
- **Hand**
  Processes the hand mask to locate and identify points of interest to build a simplified model of a hand.
- **Gesture**
  Represents a configuration of the hand that corresponds to a defined gesture.
- **GestureRecognizer**
  Analyzes the model of the hand against a set of defined gestures to return possible matches.

4.2 Capturing the frames.

After the Kinect sensor is initialized, a reader is created to retrieve multi-source frames. Since the robot will be submerged underwater, some frame sources become less effective for our implementation. For example, color frames will be very dark, where silhouette and background are almost indistinguishable and infrared frames will be distorted due to different propagation speed of the IR rays. Therefore, we will be capturing depth frames (which Kinect implements using measuring response time of laser beams) and skeleton tracking data, based on them. Another advantage of such an approach is an opportunity to swap Kinect for any other compatible device. Given any device to capture depth (could be a sonar or calibrated stereo camera) and a framework to track body joints (OpenNI, for example, is adaptable to almost any device), the following implementation can be adapted to work in these conditions with little changes to the main algorithms.

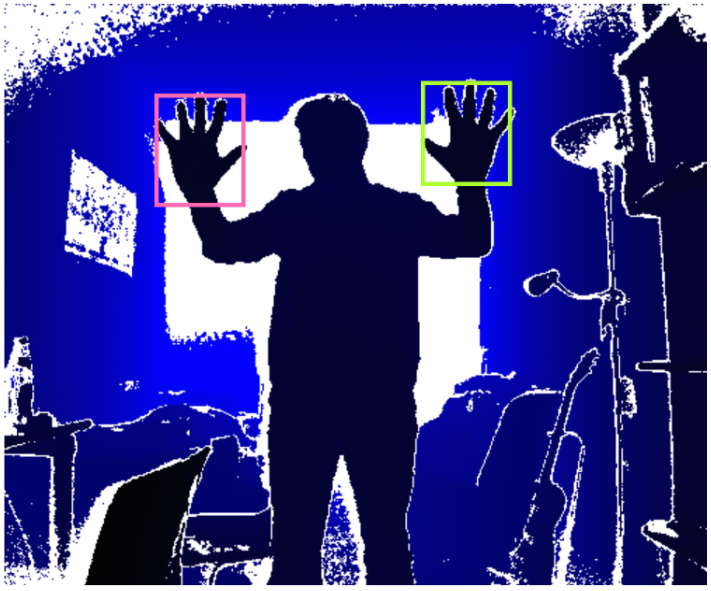

Figure 5: Depth frame returned by Kinect

After the frames are captured, the raw depth data in retrieved as *ushort[]* with distances

in millimeters to every pixels of a frame, ranging from 0 to 8000. Those values will be filtered to only make use of reliable values returned by sensor, which usually are within the range of 500 - 4500. After that, the body joints returned by the body frame will be synchronized to depth frame's 2D coordinate space, using the provided *Kinect.CoordinateMapper* class.

4.3 Hand segmentation

As a next step, we will analyze a pixel neighborhood, corresponding to hand joints, returned by the sensor. If those pixels are not a part of the body (in the other words, are in the background), we can judge that Kinect didn't return the right values for this frame and, therefore, can skip all the further processing. In the other case, a modified Scanline FloodFill algorithm will be used to segment and draw the binary mask of the hand. This particular version of algorithm was chosen for its superior efficiency, analyzing the lines of an image as opposed to individual pixels, resulting in much smaller queue size and thus, less iterations.

Instead of using a generic threshold in both directions, two separate thresholds will be used: one large one to apply to values closer than the hand's center (to ensure capturing of all the possible hand positions) and a smaller one to analyze values further away. That can be used only based on our previous assumption, that the hand is the closest object in sensor, allowing us to assume that no objects will interfere with forward thresholding. Additional check is implemented to check the distance from the hand's Z coordinate to the corresponding shoulder's Z coordinate (or head, if not available), that is aimed to minimize the number of cases, where the backwards threshold is large enough to segment bigger regions of the body, returning the data not suitable for proper gesture recognition.

In the algorithm, which detailed in the *Appendix* section, additional check (denoted with *BeyondWrist*) is made to crop values behind the wrist line. That is performed by disregarding all the pixels, projections of which on the plane, described by $v = (handX - wristX, handY - wristY)$, as returned by the Kinect sensor, are below the wrist point. However, due to noise in processing such a small region, often mistakes occur where the wrist point jumps and some fingers are being mistakenly cropped. That's why the current implementation doesn't rely on this and instead attempts to achieve different means of classifying the wrist region later on. Should a special identifying marker be allowed or additional calibration of Kinect to return more precise coordinates, this method would return a very accurate representation of the wrist line.

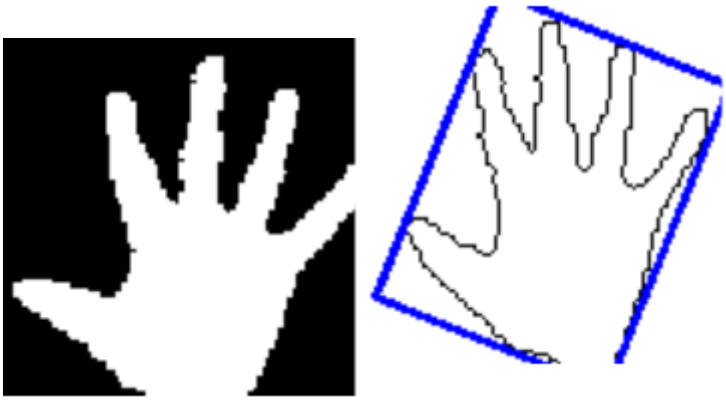

Figure 6: Hand mask, its contour and minimum area rectangle

Resulting region then will be resized to a constant size (defined as 64x64 in my implementation) to remove variance based on hand's size and orientation of the sides of the bounding box (as suggested by [8]). Its contour then is computed to find a rotated minimum area rectangle, that embeds the hand. Since the hand segmentation and following gesture recognition can take a lot of CPU, it will be done on the background thread to reduce the interference with rendering of the UI.

4.4 Palm segmentation

Before identifying points of interest on our segmented binary image, let's consider which degrees of freedom does the hand allow. On the image below (taken from [9]), we can see the general model of hand, as well as all its 26 degrees of freedom. While this model is mostly accurate, there are some people who might anatomically have an extra degree of freedom in their thumb. We will not consider such cases and instead focus on simplified version of the general model with bending angles achievable for an average person. Furthermore, to accommodate the recognition of our simple gesture set, we will consider the finger states as binary (on/off), ignoring the possible bends at the midpoints of the fingers.

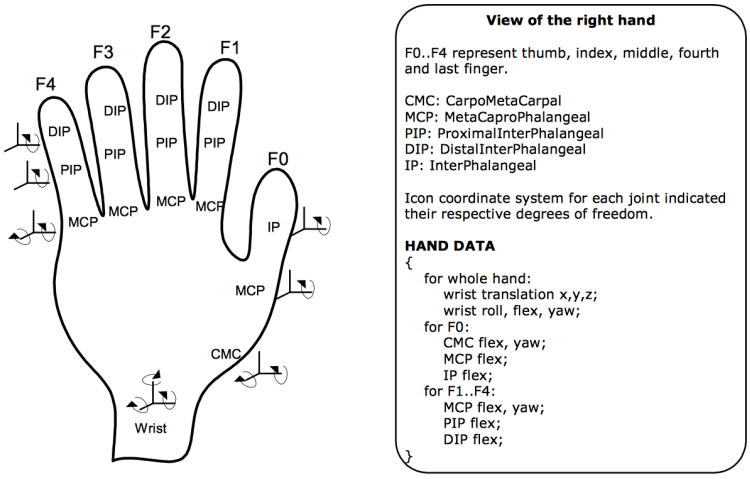

Figure 7: General model of the hand

Next step is to locate so-called ”palm point”, the center point of the palm, which will help us to identify the palm, segment it and determine approximate orientation of the hand. To do that, we will use an optimized 2D distance transform algorithm (as derived by [10], available in the *Appendix* section), where the matrix of the same size as the mask is filled with distances to the closest pixel boundary. The point with $d[i] = max(d)$ is defined to be a palm point.

Now, let's draw a circle around this point, increasing radius by one every time until non-skin values appear. Such a circle with a maximal radius will encompass the entire palm and is called inner circle. Once we've determined that, we can draw another circle, with the radius larger than inner circle's. Experimental factor of 1.5 was selected to separate the fingers, but retain the thumb even at its larger angle. This larger circle will function as a great mask, allowing us to analyze pixels outside of it for potential finger candidates.

The result of palm segmentation is illustrated at the image below:

Figure 8: Palm center, inner and outer circles on the hand contour

4.5 Finger recognition

To find the regions, corresponding to fingers, we will sample the values (360 for the best precision), located on the larger circle. If the pixel belongs to the hand of the mask, a connected region will be analyzed using a simplified version of Flood Fill to yield a region, occupied by the finger. At this point, regions smaller than the specified area cutoff (defined experimentally to be 20 pixels), will be discarded as noise to avoid detecting extra fingers. It's possible then to traverse presumed finger regions to find a fingertip (which is anatomically defined to be the point, furthest away for the center of the palm), as well as directional vector pointing from the base to that point (representing a simplified model of a finger).

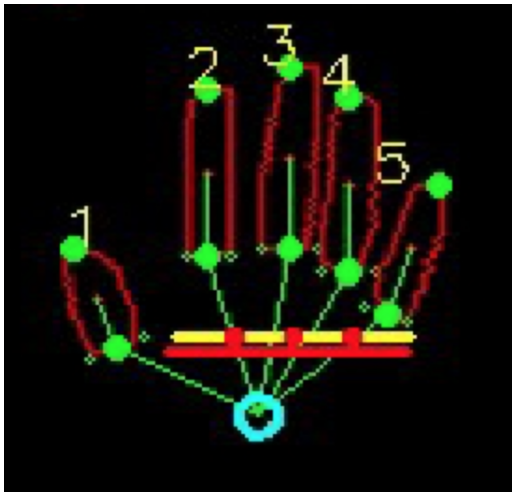

Figure 9: Model after finger recognition

However, in some of the cases this will not be enough to make a correct judgement of a finger. If fingers are grouped together (located very close to each other), one bigger region will be returned, encompassing multiple fingers. In that case, rotated minimum area rectangle will be analyzed to make a prediction about a number of fingers inside the region. Let's define base width of that rectangle to be a length of a side, completely bordering with the outer circle of the hand. Now we can compare that value with a radius of the inner hand circle (representing the width of a segmented palm), decreased by 10%, making it roughly the same as the width of 4 connected fingers. This computation will yield the following ratio (denoted λ):

$$\lambda = \frac{BaseWidth * 4}{InnerCircleRadius * 0.9}$$

The values of this ratio can be analyzed, using the robust constants as suggested by Vitali Henne [11] to make the final approximation regarding the number of fingers in the region.

- $\lambda < 1.4$: 1 finger in the region
- $1.4 <= \lambda < 2.4$: 2 fingers in the region

- $2.4 <= \lambda < 3.4$: 3 fingers in the region
- $3.4 <= \lambda < 4.4$: 4 fingers in the region
- $\lambda >= 4.4$: Discard the case as anatomically impossible.

4.6 Finger classification

After we have a list of segmented finger regions and their basic properties, we can attempt to identify the displayed fingers to use later in the gesture recognition. This proved as a very non-trivial task, considering multitude of states hand can be in and its orientation. Therefore, we have to consider the identification methods, that are invariant to the hand's rotation and number (and position) of fingers present.

Current research indicates that its extremely challenging to find two points that will yield a reliable vector in every case. However, the idea would be to draw a major principal axis, a line going from the wrist to the tip of the middle finger, that will separate the hand mask in two based on the "pixel mass" distribution. Then, we can calculate an angle of a vector from fingertip to the palm center with respect to this line. We will notice that all the fingers, but thumb will lie within a 45 degree range, allowing us to accurately select the thumb, as well as filter all the other regions, that might have been accidentally classified as fingers. Once we identify the position of the thumb, we can process the remaining figures based on their position with respect to the thumb and offset on the line, parallel to the finger base, drawn through the center of inner circle.

(Perfecting the algorithm used in this section is still in works, and will probably change by the time of the final submission).

4.7 Recognizing static gestures

Now that we can detect and identify individual fingers, static gestures can be recognized using a simple rule classifier. In our case, the hand gesture can be predicted, based on the number and indexes of the visible fingers. For example, if two fingers, index and middle one, are visible, gesture **s8** will be recognized.

In some of the cases (like **s1**/**s2** and **s5**/**s6**), the direction vector of a finger will be determined with regards to the inner circle, to determine which direction is the finger pointing to and choose the respective action. Additionally, since both hands are being tracked at the same time, more complex gestures can be defined using the combination of two hands (here, **s9**).

4.8 Recognizing dynamic gestures

Dynamic gestures are built upon the methods used to recognize static gestures, however it is not enough to describe the motion. Therefore, each particular dynamic gesture will

have its own detection function, based on the number of applicable parameters.

Simple state machine can be used to track the progress of the gesture. We start by recognizing a static hand gesture and checking whether it corresponds to the initial (starting) state of any of the dynamic gestures. If it does, the state of the machine is changed to "listening", which analyzes next consecutive frames against the requirements for a gesture . For the gestures **d1** and **d2**, for example, the global offset of the hand from the elbow will be monitored. If no movement of a sufficient offset was detected before timeout t, the corresponding static gesture (if any) will be executed. However, if hand movement is detected, a timer for the minimum duration of the gesture is setup and hand position is continued to be analyzed. If back-and-forth movement continues with defined offsets, a dynamic gesture is recognized. Based on the intensity of the movement (which can be calculated using deltas and timestamps of the frames), priority is being assigned to the task.

Some of the dynamic gestures, however, don't require moving hands. In our gesture set, **d3** will consist of no movement, but rapid changing of hand expressions. In this case, similarly, after recognizing the initial hand expression, the function will check whether the static gesture state has altered within some timeout t. Choosing sufficiently small timeout value will allow our robot to detect gestures in general without critical pauses in between, but watch out for the dynamics of the gesture as well.

5 Evaluation.

Evaluation criteria for this research project are strongly coupled with the project requirements and implementation quality. The project is assessed against the following criteria, that can be used to evaluate the results of this thesis:

The following evaluations are based on the current state of the project (as of 04/03/2015). Until the final submission, algorithms will be modified to be more robust and allow for better experimental results.

- **Functionality.**
  Application successfully detects the fingers, displayed in a frame, invariant of the hand rotation and the presence of multiple fingers in the region. Furthermore, a basic classification is applied that aims to detect the thumb, among the fingers, to correctly continue with identification. However, due to the extreme noise in the data, provided by the Kinect sensor, the reliability of this is yet to be improved (by finding a reliable vector that can be used as a basis for angle calculation). Therefore, gesture recognition works only in the "perfect" cases and will be improved to be completely robust in the following research.
- **Correctness.**
  10 test subjects of different gender and varying hand features were asked to use the application to analyze it for correctness. Experiment produced the following results:
  - Hand is being correctly segmented in 100% of the applicable cases.

  - Hand is being correctly ignored, based on all of the conditions (frontmost object in the frame, far enough from the body). The only case, that failed for 4 out of 10 participants, was the one, where user would be standing sideways (which is not a defined use case for our application). Then, shoulder Z coordinate would not be a reliable reference anymore, causing a bigger region (the entire arm) to be segmented. This can be improved by analyzing more joint values to detect the incorrect position of user with respect to the sensor.
  - Fingers were correctly located in 100% of the cases, where hand was positioned normally (even when there were multiple fingers within a region). After testing with variant hand orientations, the success rate slightly drops to 70%, based on accurateness of Kinect joint data. A better wrist approximation would allow to immediately fix all of this cases, detecting the hand in every situation.
  - Static gestures were correctly recognized only in those cases, where hand was positioned perfectly and Kinect located the thumb point successfully, which results to about 5-10% success rate. Once we will achieve the complete invariance, that was described above, expected success rate is 80-85%, based on the inaccuracies in Kinect data and hardly recognizable (distorted) hand positions.
  - Dynamic gestures were not implemented yet, since all the work has been focused on improving the results of the previous steps. However, sampling points from various Kinect joints proved the correctness of the proposed algorithm in 95% of test cases for every participant.

- **Modularity.**
  Adding new gestures to the application is trivially simple. A new Gesture object is created, describing the set of features needed to recognize the gesture and the desired callback. This object is later added to an array, maintained by GestureRecognizer to immediately enable it for processing.
  Similar approach is used with dynamic gestures, except the additional step, which is overwriting the *Evaluate()* function, that will implement the basic state machine and monitor the position of the hand across successive frames.

- **Code Quality and Documentation.**
  Class structure presented above proves a very efficient way of separating this application, where each class has its own distinctive purpose and scope. Variables and methods have descriptive names, and overall C# code design guidelines were enforced (checked using JetBrains' plugin, called Resharper).
  Writing proper documentation is still a work-in-progress, but on the moment of final submission, there will be a complete generated HTML documentation included, describing every method, class variables and class completely (similar to Doxygen style). Furthermore, all details of the algorithms which were not intuitive have a short descriptions next to them, allowing people who build on top of this code to easily understand and tweak the working mechanics of this application.

- **Testing.**
  Over the course of development I was adhering to **test-driven development** to minimize the time spent on fixing bugs and narrowing the source of potential problems. Small unit tester was written for every component of the application to assess the result of the operations we are relying on (for example, initializing the Kinect sensor)

and implemented complex algorithms (like DT and FloodFill). That allowed me to be confident in the algorithms' correctness and instead focus on filtering the noise in sensor's measurements and input data. All the actual results were tested using a live demo to account for environment noise (application works perfectly when seeded with correct hand masks).

6 Discussion.

6.1 Improving the results.

Given the precise skeletal tracking, which sadly is not completely reliable with the Kinect SDK, much better approximation of the wrist line and location of the fingers can be made, skipping a lot of computation that we are doing to calculate the trusted points. Furthermore, if a state of a thumb is returned, we can make immediate assumptions about the state of the hand, again, simplifying the recognition a lot.

6.2 Extending the application for more complex cases.

Depth mask of the hand can be reused to calculate the differences in depth at midpoints and towards the fingertip to yield a better approximation of the general model of the hand. That will allow to define more features to describe the gesture and thus, expand the recognition far beyond the simple gesture set. Multiple sources familiar with gesture recognition recommend using Hidden Markov Models for such cases to define a machine learning algorithm that automatically gathers a set of relevant features from all of the available data. Application could then be extended to include a ”learning mode”, where user can add a new gesture to the set of available ones simply by training the algorithm on the number of applicable cases under different environment conditions.

7 Conclusion.

Summarizing all the information presented above, we can see that gesture recognition, despite being a pretty novel technology, has a lot of useful applications. In our case, communicating with robot underwater can be complicated due to a number of natural events. Gesture recognition system allows to control the robot from the safe distance and keep track of health conditions of the diver. Step-by-step implementation of gesture recognition was presented, starting from hand segmentation and ending with identifying different kinds of gestures. It proved quite reliable in specific cases and further work is being done to improve to be as robust as possible.

8 Acknowledgements.

I would like to thank all the people, who helped me to succeed with implementing this guided research project:

- My guided research instructor, Prof. Andreas Birk for the advice and assistance in the course of working on this thesis.
- Prof. Horst Hahn for his excellent course in image processing, that I attended.
- Alexandru Barbarosie for the invaluable advice in geometrical analysis of the hand features and filtering the noisy data out.
- Microsoft for providing me with Kinect v2 sensor to keep after my internship.
- Finally, there is no way I would reach where I am now without my family and friends - I thank them for their support and for making me the person, that I am today.

9 Appendix

9.1 FloodFill implementation

```
FloodFillCheck (x, y, startDepth, mask) : bool
     if (!IsValidPoint(x, y)) return false
     if (depth[x, y] is not reliable) return false
     if (mask[x, y] is already marked) return false
     if (depth[x, y] > startDepth + bwdThreshold) return false
     if (depth[x, y] < startDepth - fwdThreshold) return false
     // if (BeyondWrist(x, y) return false (!)
     return true

FloodFill (startPoint, startDepth) : Hand
     Queue q, Bool[] mask
     minX = minY = +Infinity
     maxX =  maxY = -Infinity

     mask[startPoint.X, startPoint.Y] = true
     q.Add(startPoint)

     while (q is not empty)
          p = q.Pop()

          w = p.X, e = p.X
          while (FloodFillCheck(w-1, y, startDepth, mask)
              w = w-1
              mask[w, p.Y] = true
              if (w < minX) minX = w
          while (FloodFillCheck(e+1, y, startDepth, mask)
              e = e+1
              mask[e, p.Y] = true
              if (e > maxX) maxX = e

         for (i in range (w, e))
              if (FloodFillCheck(i, y-1, startDepth, mask)
                  if (y-1 < minY) minY = y-1
                  q.Push(i, y-1)
              if (FloodFillCheck(i, y+1, startDepth, mask)
                  if (y+1 > maxY) maxY = y+1
                  q.Push(i, y+1)

     position = Rectangle(minX, minY, maxX - minX, maxY - minY)
     return Hand(position, mask)
```

10 References

[1] Seong-Whan Lee. Automatic gesture recognition for intelligent human-robot interaction. Automatic Face and Gesture Recognition, 2006. FGR 2006. 7th International Conference, April 2006.

[2] Paul; Stribling Penny; Dautenhahn Kerstin Robins, Ben; Dickerson. Robot-mediated joint attention in children with autism: A case study in robot-human interaction. Interaction Studies, Volume 5, Number 2, 2004.

[3] Cory D. Kiddb Neal Lesha Charles Richa Candace L. Sidnera, Christopher Leea. Explorations in engagement for humans and robots. Artificial Intelligence, Volume 166, Issues 12, August 2005.

[4] Samsung. Smart TV Gesture Book. http://www.samsung.com/global/microsite/tv/common/guide_book_5p_vi/thumb_up.html.

[5] Microsoft Research. Kinect Sign Language Translator. http://research.microsoft.com/en-us/collaboration/stories/kinect-sign-language-translator.aspx.

[6] Microsoft Research. Kinect Launches a Surgical Revolution. http://research.microsoft.com/en-us/news/features/touchlesssurgery-060712.aspx.

[7] Marconi L. Chiarella D., Cutugno P. CADDIAN: A Human-Robot Interaction Language Based On Gestures. Draft. CNR-ILC, September 2014.

[8] Jianning Liang Jing Zhang Zhi-hua Chen, Jung-Tae Kim and Yu-Bo Yuan. Real-Time Hand Gesture Recognition Using Finger Segmentation. East China University of Technology, April 2014.

[9] Rob Audenaerde. Towards Ubiquitous Document Scanning. Master's thesis, University of Twente, December 2004.

[10] Daniel P. Huttenlocher Pedro F. Felzenszwalb. Distance Transforms of Sampled Functions. Cornell Computing and Information Science TR2004-1963, 2004.

[11] Vitali Henne. Hand Gesture Recognition with a Depth-Sensing Camera. Bachelor's Thesis, Karlsruhe Institute Of Technology, June 2011.