Towards an Autonomous Sailing Trainer

Ella Bolshinsky
Towards an Autonomous Sailing Trainer

Project Thesis

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Ella Bolshinsky

Submitted to the Senate of the Technion — Israel Institute of Technology
Nisan 5775 Haifa March 2015
This research was carried out under the supervision of Prof. Roy Friedman, in the Faculty of Computer Science.

ACKNOWLEDGEMENTS

I would like to thank my advisor, my family and everyone who has consulted or helped me in this work. More specifically, Dr. Tamar Avraham, Prof. Michael Lindenbaum, Omer Geiger, Prof. Shaul Markovitch. In addition, I would like to thank Dr. Gil Manor for his help in all the sailing related issues and Jonathan Friedman for sailing and providing us with information we can work with.

# Contents

List of Figures

Abstract 1

1 Introduction 3
   1.1 Contribution ................................................. 3

2 Short Sailing Background 5
   2.1 Instructions .................................................. 6

3 System Overview 9
   3.1 Application .................................................. 9

4 Input 11
   4.1 Input From the Side Cameras ................................. 11
   4.2 Input From the Upper Camera ................................. 13
   4.3 Input From the Smartphone Sensors ......................... 13
   4.4 Simplifying Assumptions .................................... 14

5 Input Processing 15
   5.1 Smartphone Sensors Data Processing ........................ 15
      5.1.1 Speed Measuring ....................................... 15
      5.1.2 Calculating Boat Roll and Azimuth Orientation ........... 15
   5.2 Image Analysis ............................................... 16
      5.2.1 Boat’s Centerline Angle Detection ....................... 16
      5.2.2 Boom Detection .......................................... 19
      5.2.3 Sailor Detection ......................................... 21
      5.2.4 Tiller Detection .......................................... 24
      5.2.5 Centerboard Detection .................................... 24
      5.2.6 Wind Indicator Detection ................................. 26
      5.2.7 Telltale Detection ........................................ 32
      5.2.8 Image Analysis Fine Tuning ............................... 34
      5.2.9 Image Analysis Light and Weather Conditions .......... 37
List of Figures

2.1 Detailed Laser dinghy description (taken from [ILC]) ............................... 7
2.2 A zoomed view of the objects of interest .................................................. 8
3.1 An overview of the system ........................................................................ 10
3.2 A screenshot of the application before activated ......................................... 10
4.1 Sensors to input ......................................................................................... 11
4.2 Side cameras’ location on the boat ............................................................. 12
4.3 An example picture taken by a side camera which is on the same side of the sail (downwind camera) ................................................................. 12
4.4 An example picture taken by a side camera which is on the opposite side of the sail (upwind camera) ................................................................. 12
4.5 Upper camera location on the boat ............................................................. 13
4.6 An example picture taken by the upper camera .......................................... 13
5.1 Image analysis flow chart ........................................................................ 16
5.2 Examples of upper images we have previously worked on .......................... 17
5.3 A template for the dinghy back after applying some processing ............... 18
5.4 Illustrating the deduction of the boat’s centerline angle based on the lines upon the boat ................................................................. 19
5.5 Illustrating the deduction of the boom slope based on the slope of the ellipse fitted upon the boom contour ......................................................... 21
5.6 Sailor position detection algorithm result ................................................ 23
5.7 Centerboard processing ............................................................................ 27
5.8 Example images, before zooming where the wind indicator is marked by a green rectangle ................................................................. 28
5.9 Example images after zooming on the wind indicator ................................ 29
5.10 Example images of the wind indicator after zooming and color transform ................................................................. 29
5.11 Wind indicator that was separated incorrectly to contours and the result after correcting the mistake ................................................................. 30
5.12 Wind indicator algorithm result in case only one contour was found ......... 31
5.13 Wind indicator processing results .............................................................. 32
5.14 Tiller angle detection accuracy percent in varying line lengths parameter of Hough Transform for lines ................................................................. 36
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.15</td>
<td>Tiller angle detection accuracy percent in varying max line gap parameter of Hough Transform for lines</td>
<td>36</td>
</tr>
<tr>
<td>5.16</td>
<td>The centerboard as it is seen in different light conditions</td>
<td>37</td>
</tr>
<tr>
<td>5.17</td>
<td>The telltales as they are seen in different light conditions</td>
<td>37</td>
</tr>
<tr>
<td>5.18</td>
<td>Upper frame lightening difference</td>
<td>37</td>
</tr>
<tr>
<td>6.1</td>
<td>Initial heuristic decision flow</td>
<td>41</td>
</tr>
<tr>
<td>6.2</td>
<td>Current heuristic decision flow</td>
<td>47</td>
</tr>
<tr>
<td>6.3</td>
<td>An example picture that was presented to a sailing coach in order to construct a “Golden Truth”</td>
<td>48</td>
</tr>
<tr>
<td>6.4</td>
<td>Evaluation of the machine learning versus the heuristics paradigms in terms of the amount of correctly classified instances.</td>
<td>49</td>
</tr>
<tr>
<td>6.5</td>
<td>Precision and recall of the different commands in different classifiers</td>
<td>50</td>
</tr>
<tr>
<td>7.1</td>
<td>Sequence Diagram of the Sailor Helper Application activation</td>
<td>56</td>
</tr>
<tr>
<td>7.2</td>
<td>Sailing Helper Class Diagram</td>
<td>57</td>
</tr>
</tbody>
</table>
Abstract

This work reports on our experience in designing and implementing a software/hardware bundle that implements a technical sailing trainer for the sailors of a Laser dinghy sailboat. The goal of such a trainer is to enable sailors, especially at the youth level, to practice in a genuine environment even when a human trainer is not available. Trying to keep the trainer simple and affordable, the bundle is composed of a smartphone and a small set of Wi-Fi enabled cameras placed in designated places on the boat. Computationally expensive tasks may be processed on a remote server. The bundle analyzes the condition of the sailboat by processing the pictures taken by the cameras and the smartphone sensors’ reading and provides instructions to the sailor based on this analysis. The recommendations are generated from a combination of heuristics and machine learning mechanisms. Some of the main challenges of the system include extreme changes in lighting and weather conditions, sea spray, slight shifts in cameras locations due to strong winds and the waves, tracking multiple moving objects, and terminating all the processing in real-time before the next set of images and sensor readings arrives. A prototype of the bundle was created and implemented. The performance and limitations of all components were studied, although due to technical barriers, we fell short of a complete live deployment. In the process, we have accumulated some interesting insights about building such low cost real-life autonomous systems.
Chapter 1

Introduction

Replacing a human coach with an autonomous system can help athletes improve their skills as it enables them to hold additional training sessions in which they get feedback about their performance and what they should do to improve it. Obviously, a human coach provides training and mental support at multiple levels, and in the near future we do not see how this can be completely replaced by a computerized system. Yet, it is interesting to explore the ability of providing an autonomous training capability at least for the technical aspects of the sport.

Specifically, in this work, we focus on sailing. The training of competitive sailors of individual sailboats, such as the Laser dinghy, typically combines technical maneuvers (boat handling and sail trimming), tactical considerations, as well as strategic planning. The former usually involves the trainer following closely the sailor on the water, shouting instructions to the sailor. However, especially at the youth level, a single trainer usually works with a group of several sailors, meaning that the trainer can only devote a fraction of a training session to each sailor. Moreover, sailors often add extra-curricular training sessions by going out to the sea without their trainer. Having a reasonably priced easy to operate bundle that can give such technical instructions could, therefore, greatly improve the effectiveness of such training sessions.

1.1 Contribution

In this work, we design a technical sailing trainer. The bundle we create should be economic in term of power consumption and battery life in order to enable a full training session. In addition, it should be relatively light weight, inexpensive, ubiquitous, non-intrusive and simple to deploy. If it weighs more than a few hundred grams, it can already impact the boat’s performance, and if we decide to place something on the sail, even a few grams can already make an impact. If the price is not affordable, young sailors will not be able to afford it. If it is intrusive or involves complex deployment, sailors would shy away from it. Another extremely important characteristic is ease of use. The sailor has to manipulate the boat and to make strategic planning, which require significant efforts. Thus, the autonomous trainer should not be a burden to the sailor.

In this dissertation, we report on our experience in building a prototype of the bundle based on the above goals. Our bundle consists of a smartphone (Samsung Galaxy S3) and three wireless cameras
(GoPro Hero 3 White) placed at designated positions on the boat. The cameras take periodic pictures of the boat (and sailor) while the phone reads its integrated sensors and the images taken by the cameras, analyzes them to determine the sailboat’s and sailor’s status and provides vocal instructions to the sailor (e.g., through BT headsets). The phone may offload some of the heavy computation tasks to a remote server.

The task of the autonomous trainer consists of two main parts: correctly analyzing the current status and then deciding on further actions. The first part employs image analysis capabilities and parallel computation of inputs arriving from different sources, while in the latter we try two paradigms: a predefined rules based expert system and a machine learning process. We compare the performance of these techniques and can combine their recommendations in the final output.

We have implemented the bundle and studied its performance and limitations. As we report in this thesis, due to some limitations we encountered in existing technology and performance, while all parts of the system have been implemented and tested, we stopped short of full live integration of all system’s parts. Yet, the results we have obtained exhibit in principle the feasibility of deploying a complete system in the very near future, once the technological bottlenecks we have identified are resolved. In the process, we have also accumulated some interesting insights about generally building such low cost real-life autonomous systems.

The work was done in collaboration with a devout sailor, Dr. Gil Manor. Gil was 6 times national champion in Laser racing, participated in multiple European and World championships, and served until recently as the sailing coach of the Haifa Sailing club’s Laser Team as well as the Israeli National Laser Team. Hence, in addition to being a researcher, Gil served as our sailing expert on board.

**Roadmap:** The rest of this thesis is organized as follows: A brief sailing terminology and goals are given in Chapter 2, including the instructions our trainer provides. Chapter 3 outlines the system overview followed by identification of the inputs we are seeking in Chapter 4. Chapter 5 describes the input processing, both from the smartphone sensors and the image analysis, followed by accuracy evaluation of the image analysis. The recommendation engine is detailed in Chapter 6. The description of the deployment, timing and power measurements appear in Chapter 7. In this chapter we present the life cycle of the application and discuss its time constraints, the systems it was tested on and the current challenges in connecting to the cameras and downloading the images. Following is the related work in Chapter 8. We conclude with a discussion in Chapter 9 and provides ideas for further work.
Chapter 2

Short Sailing Background

In this work, we concentrate on the Laser dinghy. This is a single handed sailboat that is also a one-design Olympic class. In this thesis, we limit ourselves to Laser Radial boat only. However, our work can be easily adjusted to other single sailor, single sail boats. This is because the differences are mostly in the boat parts’ dimensions. Thus, only some technical adjustments should be done in order to work with such boats (for instance the location of the cameras might be changed).

We briefly describe objects of interest on the sailboat and the kind of instructions that can be provided with relation to them. Obviously, our goal is not to teach the reader how to sail, but rather to provide some common terminology to understand the rest of the thesis. Specifically, in this work we address only the parts of the boat that are relevant to our sailing trainer and include (see Figures 2.2a and 2.2b):

1. **Boom** - A horizontal beam located at the bottom of the sail [And03].

2. **Tiller** - Used by the sailor to steer the boat.

3. **Cockpit** - An area on the boat surface where the sailor usually sits.

4. **Sheets** - The Laser contains several sheets that are used to shape the sail and to determine the boom position. For our work, we do not distinguish between the different ropes.

5. **Telltales** - Short threads that are attached to the sail and serve as indicators for the airflow over the sail. There are several telltales on the sail. However, in this work we only address two of them (attached from each side of the sail). We refer to the colored circle that attaches the telltale to the sail as the telltale base.

6. **Centerboard** - A board placed in the hull (the main body) of the sailboat and provides a counter force to possible currents and the lift force that pushes the sail (and thus the boat) to the side. The counter force provided is dependant on the percentage of the centerboard located in the water.

7. **Wind indicator** - Points to the direction of the wind.

The telltale, the wind indicator and the centerboard are marked in Figure 2.2a while the boom, the tiller and cockpit in Figure 2.2b.
2.1 Instructions

The trainer attempts to imitate a human trainer by supplying remarks to the sailor. These remarks are generated in response to the inputs the trainer acquires. The possible set of instructions was specified by Gil as well as collected by observing training sessions. It is important to note that our autonomous sailor trainer does not cover all the possible instructions observed in the training sessions. For example, in this work we do not address wind gusts. Possible remarks and instructions that were integrated in the system and their meaning are described in Table 2.1.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>What should the sailor do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flatten the boat</td>
<td>Change boat roll angle so that it will be smaller than 15 degrees</td>
</tr>
<tr>
<td>Move to the front</td>
<td>Change his location to the front of the cockpit</td>
</tr>
<tr>
<td>Loose the sail</td>
<td>Loose the sail</td>
</tr>
<tr>
<td>Tight the sail</td>
<td>Tight the sail</td>
</tr>
<tr>
<td>Push the tiller</td>
<td>Push the tiller away</td>
</tr>
<tr>
<td>Ease the ropes</td>
<td>Ease the different ropes on the boat</td>
</tr>
<tr>
<td>Take centerboard out</td>
<td>Raise the centerboard</td>
</tr>
<tr>
<td>Put centerboard in</td>
<td>Lower the centerboard</td>
</tr>
<tr>
<td>Use the tiller less</td>
<td>Make less changes with the tiller</td>
</tr>
</tbody>
</table>

Table 2.1: Trainer Instructions Table
Parts of the Laser

- Length overall: 4.23 m
- Length waterline: 3.81 m
- Beam: 1.37 m
- Sail area:
  - Laser Standard: 7.06 m²
  - Laser Radial: 5.76 m²
  - Laser 4.7: 4.7 m²
- Nominal weight (with fittings): 58 kg
- Positive flotation: 158.7 kg

Designed by Bruce Kirby

Figure 2.1: Detailed Laser dinghy description (taken from [ILC])
Figure 2.2: A zoomed view of the objects of interest

(a) The sailboat view from its side camera

(b) A zoomed view of the sailboat from the upper camera
Chapter 3

System Overview

We decided to minimize the number of devices used in order to make the system as lightweight as possible, utilizing only external cameras and an Android smartphone. In principal, other external sensors could be used. For instance, sensors presented in [Cod08] could be used in order to measure the pressure on the sail of the sailboat and an external anemometer could measure the wind direction. However, in order to make this system as user friendly as possible, we decided to minimize the number of devices needed for the system.

The cameras used for our research are three GoPro HERO3 White Edition cameras, with a resolution of 5MP. These cameras enable us to shoot a photo once in two seconds (and even less) which is a sufficient frequency for our system. In addition, these cameras are lightweight and small. The application is installed on a Samsung I9300 Galaxy S3 smartphone. This phone has a Quad-core 1.4 GHz Cortex-A9 CPU and has a Mali-400MP4 GPU. The operation system on the phone is Android 4.0 (Ice Cream Sandwich). It also contains GPS and the following sensors: Accelerometer, RGB light, Digital compass, Proximity, Gyro and a Barometer.

We had not known in advance what would run on the phone and what will require processing by a more powerful computer. An overview of the system design is depicted in Figure 3.1. It consists of a smartphone and external cameras that get the input for the system. This input is processed either on the phone or partially on a remote server (or the cloud). In the latter case, the processing results return to the smartphone, enter a recommendation engine and eventually a notification is read vocally to the sailor.

3.1 Application

As the sailor should concentrate on the sailing and not on operating any applications on his smartphone, the application is extremely simple. It contains a single screen with bold “Start” and “Stop” buttons a frame in which the speed and the compass reading are written. An option to preview the pictures from the cameras is also available to make sure that the cameras are well located. This button downloads the first available picture from each camera upon each activation and at last presents a compass picture that points to the north. The instructions given to the sailor are both written on the screen and if it is not “You are doing well!”, are also read aloud, so that the sailor does not need to look at the application.

Figure 3.2 illustrates the application when it is stopped. Thus, in the instructions’ frame on the screen
“Stopped!!!” is written. The application also maintains several logs. One log records the compass, speed and orientation reading. Another one records the times of pressing the “start” button and the commands sent to the cameras. We used this log in order to match the taken pictures to the readings from the first log, while processing the reading offline.
Chapter 4

Input

Figure 4.1 depicts the indicators about the boat and sailor we need in order to generate meaningful instructions. In particular, the input of the autonomous sailing trainer contains information from cameras installed on the boat, as well as the smartphone sensors.

![Sensors to input diagram]

4.1 Input From the Side Cameras

Two side cameras are located on the boat (from both sides of the boat), as illustrated in Figure 4.2. With both cameras we can deduce the location of the centerboard. The camera that is located on the same side as the sail is used to determine the Telltale status. As the sail can be on either side of the boat, one side camera is not sufficient. The telltale is sometimes seen in the other camera as well, as there are telltales from both sides of the sail. However, the one that interests the sailing trainer is the one seen by the camera on the same side as the sail (the down-wind camera). This side camera also serves to detect
the wind indicator. It is much easier to detect the wind indicator from this side camera as in this case the wind indicator is mostly on the sky background, while on the other camera it is often in front of the mast and the sail, causing a lot of distortions.

Figure 4.3: An example picture taken by a side camera which is on the same side of the sail (downwind camera)

Figure 4.4: An example picture taken by a side camera which is on the opposite side of the sail (upwind camera)
4.2 Input From the Upper Camera

The upper camera is located on the mast, as illustrated in Figure 4.5. Via the upper camera we can detect the boom, the tiller and the sailor.

4.3 Input From the Smartphone Sensors

As already mentioned, we rely on the smartphone sensors. More specifically, we employ the accelerometer, magnetic field and GPS sensors. With the help of the sensors we check the roll of the boat, its azimuth (in order to deduce the compass reading) and its speed. Accessing the sensors is made by registering via a SensorManager Android class, and getting notifications about changes in the sensor’s measurement.
4.4 Simplifying Assumptions

In this work we make several assumptions. First, in order to make image analysis results more accurate and ease the processing, we mark the following parts of the sailboat: the end of the boom is marked with a red isolation band and red lines are placed on the boat surface around the cockpit. These lines continue to the edges of the boat and a single red line is also drawn to mark the back of the boat (stern). In addition, in order to identify objects on the sailboat we make some assumptions about their color. We assume the telltale is either red or green (depending on the sail side) and its base is a circle of the same color (might look like an ellipse in the pictures). Sailor’s shirt is assumed to be blue and the tiller is assumed to be black while the boat and the sail are white and the centerboard is yellow.

We assume the sailor sits in the opposite side of the boom, as this is usually the case, and it is very infrequent for him to sit otherwise (this mainly happens during tacking or jibing, when the sailor shifts sides). In addition, the legs of the sailor are always assumed to be inside the cockpit and not outside of it, as this is the normal sailing behaviour (we do not address the cases when the sailor fixes something on the sailboat or leaves the cockpit for any other reason). For instance, in Figure 4.6, the sailor sits in the opposite side of the boom and with his legs inside the cockpit, as we would expect.
Chapter 5

Input Processing

5.1 Smartphone Sensors Data Processing

5.1.1 Speed Measuring

In order to measure the speed of the dinghy over the water we employ Android LocationManager class. LocationManager contains three location providers: GPS provider, network provider and passive provider. Network provider determines location based on availability of cell tower and WiFi access points while the passive provider passively receives location updates when other applications or services request them without actually requesting the locations itself [And]. GPS provider gave a sufficient accuracy as we are located on an open area. However, as it is expensive in terms of battery usage we tried the other providers and tried to use the accelerometer in order to obtain less frequent updates from the providers. As the accelerometer contains also the gravity, we employed the Android sensor, TYPE_LINEAR_ACCELERATION which is a "synthetic" sensor providing abstraction over the raw acceleration sensor and yields high-pass-filtered accelerometer data, removing the gravity force. If one wants to get the acceleration while excluding the gravity, it is preferable to use this sensor rather than filtering within the application [GM]. Unfortunately, all our power saving attempts did not yield a sufficient accuracy and we kept using the GPS only.

5.1.2 Calculating Boat Roll and Azimuth Orientation

We want to check the roll orientation of the boat via the smartphone sensors. For checking this parameter, the phone should be placed on the surface of the boat, when the phone top is pointing to the boat front. The sensors we use are accelerometer and magnetic field. In order to track the changes, we register to both sensors and upon a change in one of the sensors, we check the extent of the change. Upon a change we update previously stored roll and azimuth by an AsyncTask in order not to delay the UI thread. First, we compute the rotation matrix $R$ creating a transformation of a vector from the device coordinate system to the world’s coordinate system. Afterwards, we remap $R$ to an x,y coordinate system. At last, we use the rotation matrix to compute the device’s orientation and extract the roll and the azimuth. The roll value is used to deduce whether the sail boat leans to some side while the azimuth is used to deduce the compass reading.
5.2 Image Analysis

At any instant we have one current image from each of the three cameras installed on the dinghy. We denote the image coming from the upper camera the *upper frame*, the one coming from the right camera the *right frame*, and the one coming from the left camera the *left frame*. Both left and right frames are also referred to as *side frames*. As noted before, we have several input parameters acquired from these cameras. Because the image analysis calculations are time consuming, we try to parallelize them as much as possible. We start by processing the upper frame in order to detect the dinghy. We calculate in parallel the boat’s centerline angle and the boom relative to the boat. The acquired information about the boat centerline angle is needed for the tiller and the sailor detection. Therefore, these two tasks start right after the boat’s centerline angle calculation has finished. The telltale detection relies on knowing the side of the sail relative to the boat center and the angle created between the boat center and the boom. For the wind indicator detection we do not need the exact angle, but we do need to know on which side is the sail relative to the boat. Thus, when both boom and boat’s centerline angle detection processes finish we can start detecting the telltale and the wind indicator. The centerboard position is obtained from both side frames and is not dependant upon any other image analysis task. Thus, it is started at the moment we get the input pictures. The flow chart of the image analysis performed by the sailing trainer software is described in Figure 5.1. We will describe the components of this chart in details in the following sub sections. In the process of image analysis we have employed the OpenCV library [BK08].

![Image Analysis Flow Chart](image_analysis_flow_chart.png)

**Figure 5.1: Image analysis flow chart**

5.2.1 Boat’s Centerline Angle Detection

Boat’s centerline angle is a significant factor as we check the tiller angle and the boom angle relative to the boat. In addition, after detecting the boat we can limit our search area in 3 of the picture processing tasks ( telltale detection, sailor detection and tiller detection). Limiting the search area achieves two goals: lowering the computation time and improving accuracy by avoiding confusion with surrounding noise. Thus, detecting the boat is an important factor in the process of image analysis done by the sailing trainer. In order to ease this task, red isolation band lines are placed on the boat and the boom end is colored in red.
Previous Attempts Which Were not Eventually Deployed

Initially, no markers were present on the boat, as depicted in Figure 5.2a. We did not work much on pictures without any markers. Therefore, we will only describe the work done after adding markers. At first, we added black markers upon the boat without marking the boom, example is Figure 5.2b.

In the presence of the black markers we detected the boat’s centerline angle and the boom in two parallel threads. While one thread was detecting boat’s centerline angle only, the other detected two intersecting lines. One line represents the boat and the other represents the boom. Afterwards, the thread results were combined to determine the boat’s centerline angle and boom’s location and slope relative to the boat.

The thread that detected the boat’s centerline angle worked in the following way: first we applied a mask in order to get only the back part of the boat. Afterwards, we applied Harris and Canny edge detectors in order to look for lines present in the picture. We looked for straight lines with Hough Transform algorithm. Upon obtaining the lines, several heuristic checks were made. For instance, how probable is the slope of the line given its location, how many lines indicate the same slope and what is the distance between them (we know the approximate distance between the lines as it is seen from the upper camera). Eventually, a selected slope was returned.

An additional method to find the boat’s centerline angle that we considered was to apply template match for the back part of the boat after applying Harris corner detector (with possible rotations of the template). A template in a single rotation angle is shown in Figure 5.3. This attempt proved to be unsuccessful, probably due to the fact that the tiller changes and as there are a lot of possible angles. In addition, sometimes the sail conceals part of this area.

In parallel to this thread, we tried to find two lines that represent the boat and the boom. In this case as well, we applied Harris edge detection on the grayed image. Afterwards, we smoothed the image and applied the Canny edge detection algorithm. Finally, Hough Transform for lines gave us the lines on the image. We know that the continuation of these lines intersect in an approximate constant location on the boat front. Therefore, we tried to find two lines whose continuations intersect approximately in this location (we checked if the intersection is located in a predefined range). We can define this intersection point range due to the constant upper camera location. Nevertheless, we cannot determine a
single intersection point as the camera moves a bit in relation to the boat (due to mast movement). After
getting all the candidate lines, we checked heuristically if the angle between pairs of lines seems to be
probable and fits the distance between the lines. In addition, we verified the distance to the intersection
point and lines’ location relatively to their intersection point. Eventually, the lines which had minimum
distance from the intersection point were selected.

After both threads terminated, we combined the results to indicate the boom and the boat’s centerline
angle. This method was not good enough as we had less than 30% detecting accuracy. Some of the errors
were due to the fact that the boom did not have any particular color and the boom was in a lot of cases
hidden by the sail. Thus, significant error stemmed from detecting lines on the sail. In addition, coloring
the lines on the boat in a unique color also aided a lot in detecting these lines and not confusing them
with other lines.

Current Process of the Boat’s Centerline Angle Detection

The process with the red isolation band markers includes cropping an area of the image. We remove
areas which the boat cannot reach, due to the upper camera location (Line 1 in Algorithm 5.1), applying
constant matrix operations upon the image in order to identify the areas which are colored in red (Line 2
in Algorithm 5.1). In this stage, we tried applying Canny algorithm to mark edges. However, it showed
worse results, thus was not applied here. Afterwards, we look for lines with Hough Transform for
lines algorithm (Line 3 in Algorithm 5.1). If no lines were detected, we try to employ a simpler matrix
operation (Line 5 in Algorithm 5.1) on the picture (subtracting green channel from the red channel) and
apply again the Hough line algorithm (Line 6 in Algorithm 5.1). After detecting some candidate lines,
we filter the irrelevant lines and keep only the lines that either define the boat’s centerline angle or are
perpendicular to the boat’s centerline angle (both are marked in red).

We go over each of the candidate lines and first check if it could define the boat’s centerline angle
(Line 11 in Algorithm 5.1). We denote these lines as the boat’s centerlines. We check if a line is a boat’s
centerlines by its position (relative to its slope) and by the angle formed from the approximate boat front
to the relevant picture edge and the line. If this angle is smaller than a predefined threshold, there is a
high probability for this line to be the boat’s centerline. If this could be a boat’s centerline we save it and

Figure 5.3: A template for the dinghy back after applying some processing
continue to the next line. Otherwise, we check if this line is perpendicular to the line that starts from the boat front (bow) and has the boat’s slope (Line 11 in Algorithm 15). We denote these lines as boat edge lines.

These lines are selected in two ways. First, we draw a line from the line center to the approximate boat front. If this new line is perpendicular to our line, our line has a good chance to be a boat edge line. This condition does not necessarily hold, especially if the line was detected only partially. Thus, we also check if the candidate line is perpendicular to some boat’s centerline. Obviously, when we say perpendicular, we do not look for exactly 90 degrees, but define some threshold. When selecting both boat’s centerlines and boat edge lines, in order to eliminate noise, if several lines imply a single line, we select it only once.

After we have boat’s centerlines and the boat edge lines, we should deduce the boat’s centerlines angle and which of the lines that were previously selected are correct. Thus, we calculate for each candidate slope a grade based on the amount of boat’s centerlines and the boat edge lines that support it and the distance between parallel boat centerlines. We select the slope with the maximal grade (Line 20 in Algorithm 5.1).

However, if we have a few candidates with the maximum grade, we try to resolve the conflicts by selecting the slope closest to the previous slope, as usually the slope does not change drastically. Once we find the boat’s centerlines angle, we also select the lines which are correct (as they support the selected slope). This helps us in limiting the search area on the next image analysis tasks.

The deduction of the boat’s centerlines based on the lines on the boat is illustrated in Figure 5.4. The green lines on this picture illustrate the lines found upon the boat, while the blue line illustrates the deduced slope.

5.2.2 Boom Detection

At first, when the boom end was not colored in red, the process of its detection was combined with the boat’s centerlines angle detection as described in 5.2.1. As previously noted, the boom is detected in the upper frame. The upper camera is located in an approximately constant location relative to the boom. Thus, first we crop the image to be as small as possible while containing the boom (Line 2 in Algorithm 5.2). Afterwards, we perform an opening transformation on the image (containing an erode operation
Algorithm 5.1 Boat’s centerline angle detection pseudo code

{Input: $Img$ image from upper camera}
{Output: The boat’s centerline angle}
1: $cropped := crop(Img)$;
2: $trans := matrixOperations(Img)$;
3: $lines := HoughLinesP(trans)$;
4: if $lines.length=0$ then
5: $trans := redSubGreen(cropped)$;
6: $lines := HoughLinesP(trans)$;
7: end if
8: $boatCenterLines := \emptyset$;
9: $boatEdgeLines := \emptyset$;
10: for each line in lines do
11: if $isLineOnBoatBySlope(line)$ then
12: add(line, boatCenterLines);
13: continue;
14: end if
15: if $isLineOnBoatEdge(line)$ then
16: add(line, boatEdgeLines);
17: end if
18: end for
19: $slopeToSupportLines := calculateSlopeToLines(boatCenterLines, boatEdgeLines)$;
20: $selectedSlope := selectSlope(slopeToSupportLines)$;
21: return $selectedSlope$
5.2.3 Sailor Detection

We would like to locate the sailor in the cockpit and determine how close he is to the cockpit’s front. We locate the sailor after detecting the boat’s centerline angle by the red isolation band attached to the boat. Therefore, we can first try to find the cockpit by the red lines indicated on the boat surface. There are three red lines on the boat that are horizontal to the boat. We previously denoted these lines by boat edge lines. Two of them define the cockpit edges. If only some of the three lines are found, we try to decide which lines were found and by the approximate distance between the lines, determine where the others should be.

First, we take all the horizontal red marker lines found on the boat and divide them into three groups. One group includes lines from the cockpit bottom, the second group is the cockpit top while the third is the edge of the boat, which is also marked in red. When mapping the lines to the three groups we consider the distance between the lines, the vertical red markers located above and under the candidate horizontal line, the distance to approximate boat center reference point and the cockpit size. The two last factors can be approximately determined due to the constant location of the upper camera. It is important to note that this knowledge gives us very approximate distances that differ a lot due to mast movements, thus we do not just define constant threshold, but try to decide where each line is located heuristically. After considering all these factors we deduce to which of the groups each line is suitable.

Our aim is to bound the cockpit edges by the two lower horizontal lines. Thus, we construct a single line out of each of the previously mapped groups. First, we construct a single line for the lower cockpit edge and the boat edge. Next, we construct the upper cockpit line, while checking that the lines that construct it are located in a reasonable distance from the previously constructed lines. If they are not, we skip those lines.

When we construct the line, if its length is not sufficient as the approximate cockpit borders length plus some additional factor (as we want to catch the sailor even if he leans outside of the boat), we make the lines a bit longer. If some of these three lines are missing (were not detected in the picture), we try to
Algorithm 5.2 Boom Detection pseudo code

{Input: Img image from upper camera}  
{Output: Boom location and slope}  
1: try := 0;  
2: cropped := crop(Img);  
3: if try == 0 then  
4:   cropped := open(cropped);  
5: end if  
6: cropped := matrixOperations(cropped);  
7: contours := findContours(trans);  
8: selectedContour := null;  
9: worseContour := null;  
10: for each contour in contours do  
11:   if (canEliminate(contour)) then  
12:     continue;  
13:   end if  
14:   if (largerThenExpected(contour)) then  
15:     if (better(contour,worse)) then  
16:       worse := contour;  
17:     end if  
18:   end if  
19:   grade := grade(contour);  
20:   if grade > maxGrade then  
21:     maxGrade := grade;  
22:     selectedContour := contour;  
23:   end if  
24: end for  
25: if selectedContour == null AND worse == null AND try == 0 then  
26:   try := try + 1;  
27:   goto 2;  
28: end if  
29: if selectedContour == null AND worse! = null then  
30:   selectedContour := worse;  
31: end if  
32: selectedBoom := extract(selectedContour);  
33: return selectedBoom
deduce their location by the approximate size of the cockpit and by the lines that are present.

After we have decided about the lines that bound the cockpit, we fill all the outside of the image in black (preserving only the cockpit in its original color). As noted in the simplifying assumptions, we assume that the sailor’s shirt is blue. Thus, we perform matrix operations over the image to identify the sailor’s shirt, look for contours in the image and select the largest contour (this is our candidate for the sailor). We calculate all of the moments up to the third order of the contour and calculate the center of the mass via the returned moments (in this case, the center of the sailors shirt). This calculation gives a good approximation of where the sailor sits.

Now, when we know the sailor’s location and the bounds of the cockpit we can calculate where in the cockpit the sailor sits. An example of the result of the algorithm is presented in 5.6 where the sailor position is 45% percent to the front of the cockpit.

**Algorithm 5.3** Sailor detection pseudo code

```plaintext
{Input: Img image from upper camera, edgeLines that represent the boat edge lines}
{Output: Sailor location inside the cockpit}
1: cropped := crop(Img);
2: sortEdges(edgeLines, lowerEdges, upperEdges, bathEdges);
3: lowerBathLine := getLine(bathEdges);
4: upperEdgeLine := getLine(upperEdges);
5: upperBathLine := getLine(lowerEdges, upperEdgeLine, lowerBathLine);
6: mask := createMask(lowerBathLine, upperBathLine);
7: img := applyMask(img, mask);
8: img := matrixOperations(img);
9: contours := findContours(img);
10: maxAreaContour := selectContours(contours);
11: moments := calcMoments(maxAreaContour);
12: center := center(moments);
13: sailorLocation := calcLocation(center, lowerBathLine, upperBathLine);
14: return sailorLocation
```
5.2.4 Tiller Detection

First, we know approximately at which radius from the dinghy front the tiller should be located. This allows us to apply a constant mask on the image before doing any further actions. By this simple operation, we narrow our search area, which, as already mentioned, saves computational time and yields better results. In addition, we detect the tiller after detecting the boat’s centerline angle by the red isolation band located on the boat. This allows us to narrow our search area even more. Thus, our second action is to crop the image according to the previously detected lines on the boat. If the resulting area is very small, for instance, if only some of the lines on the boat were detected correctly, we enlarge the area to some minimum size. In addition, the front of the boat is located in a rather constant location in the picture. Therefore, we also check that the area is in a reasonable distance from this location. If it is not, we adjust it by moving the area and changing its size.

Afterwards, we perform an Opening transformation on the image (containing an erode operation followed by a dilate operation). This removes small irrelevant objects. Due to the specific color of the tiller we perform matrix operations upon the image to identify the areas with the specific color. As the tiller is a straight line, we use Hough Transform for lines to detect it. Prior to detecting the lines, we apply Canny edge detection algorithm on the image.

Once we have the lines, we go over the lines and calculate the slope for each of the lines. If the line is too close to the boom and can be merged with it, we skip it. In addition, if the slope of the tiller is too big, we skip it.

Afterwards, we check how many white dots surround the line and how many are located on each side of the tiller candidate. The reason for this check is that the tiller is located on a white boat. By performing this check we split the lines into 3 groups, most probable, less probable and the least probable. We take the result from the first group which is not empty. If all lines in this group agree upon the slope we select it. We take into account whether the slopes are large or small when checking if they can be merged, allowing a larger delta between bigger slopes. The merged slope is the average of the given slopes. If the slopes cannot be merged, we resolve the conflict.

As previously noted, we first detect the red lines marked on the boat. When resolving the conflict, first we remove lines that are not in the edge of the boat (have lines above them on the boat). In addition, we calculate for each tiller candidate on what distance from the lines on the boat it is located, and if it has lines on the boat around it (on which sides). In this case as well, we split the candidates to three groups with increasing probability. The result is selected from the most probable group which is not empty. If conflicts still exist, we merge the slopes that can be merged, and select the most supported slope. We also take into account the length of the candidate tiller and reject less probable slopes.

5.2.5 Centerboard Detection

We should measure the extent in which the centerboard is seen above the boat. This allows us to deduce the existing contra force to the force that pushes the sail aside. The centerboard is seen on both side frames. Thus, we process the side frames in parallel to find the centerboard location with a higher confidence. As the side cameras are attached to the boat surface and the centerboard is attached to the hull, the centerboard location relative to the camera can be limited to a small area. This, as usual, allows us to compute the
Algorithm 5.4 Tiller detection pseudo code

{Input: Input Img image from upper camera, boatSlope as previously calculated and the lines over the boat which were previously detected correctEdgeLines and correctBoatCenterLines} 
{Output: Tiller slope}

1: Img := mask(Img);
2: cropped := crop(cropped);
3: cropped := adjust(cropped);
4: cropped := open(cropped);
5: cropped := matrixOperations(cropped);
6: cropped := Canny(cropped);
7: lines := HoughLines(cropped);
8: mostProbable := ∅;
9: mediumProbability := ∅;
10: leastProbable := ∅;
11: for each line in lines do
12:     lineSlope := slope(line)
13:     if notprobable(lineSlope, boatSlope) then
14:         continue;
15:     end if
16:     soroundingWhite := soroundingWhite(line);
17:     addToGroup(line, lineSlope, soroundingWhite, mostProbable, mediumProbability, leastProbable);
18: end for
19: selected := selectFirstNotEmpty(mostProbable, mediumProbability, leastProbable);
20: return deduceSlope(selected, correctEdgeLines, correctBoatCenterLines)
21:
22:
23: Function deduceSlope(list candidateLines, list correctEdgeLines, list correctBoatCenterLines)
24: if (canMergeSlopes(candidateLines)) then
25:     return mergedSlope(candidateLines)
26: end if
27: mostProbable := ∅;
28: mediumProbability := ∅;
29: leastProbable := ∅;
30: for each line in candidateLines do
31:     dis := calcDistance(candidateLines, dis, relation, mostProbable, mediumProbability, leastProbable);
32:     relation := checkRelation(candidateLines, correctEdgeLines, correctBoatCenterLines);
33:     addToGroup(line, dis, relation, mostProbable, mediumProbability, leastProbable);
34: end for
35: selected := selectFirstNotEmpty(mostProbable, mediumProbability, leastProbable);
36: return mostSupportedMergedSlope(selected);
37: EndFunction
result faster and more precisely and, therefore, we first crop the image according to a predefined area (Line 1 in Algorithm 5.5). Afterwards, we smooth the image by applying Gaussian smoothing to eliminate noise (Line 2 in Algorithm 5.5). Previously, we tried to apply opening morphological transformation. However, this has yielded worse results as this smoothed the picture too much.

We know the color of the centerboard is yellow. Therefore, we apply constant matrix operations to identify this color (Line 3 in Algorithm 5.5). The yellow color is highly influenced by the lightening as we will further explain in section 5.2.9. Thus, when performing the transformation we take the lightening into account. Afterwards, we find the contours that have the specific color we identified (Line 4 in Algorithm 5.5). If no contours were found the centerboard is assumed to be in the water.

If we do find relevant contours, we go over them. For each contour, if its size and shape or its position are not probable, we disregard it. Otherwise, we try to detect the borders (maximal and minimal x and y values) of the found contours. In addition, during this iteration we sum the areas of the found contours and we hold the maximal height of the found contour. Additional factors we check are the contour which has the largest y coordinate and the second highest contour (by the same matric). If the gap between those contours is too large and the highest contour is not very big, we subtract the gap between them from the returned height.

We try to estimate the accuracy of our result. We will use the accuracy parameter when integrating the results from both side frames upon resolving conflicts. Therefore, we initialize a confidence parameter and decrease it in the following cases: the width of the centerboard is too wide or too narrow, the summed area is much smaller than the calculated centerboard area by the found X and Y limits, or much smaller than the height found by the limits multiplied by a constant factor. If the last condition holds, we assume that only the biggest contour found is relevant and the rest are noise and return the stored maximal height.

Figure 5.7 illustrates the main phases of the centerboard processing. The image is cropped and afterwards we perform some matrix operation to isolate the desired color. The white rectangles in the last frame represent the rectangles that bound the contours and define the found centerboard parts. While, the red lines on the last frame limit the search area on the frame. The returned length is the total height from the lowest point of these rectangles to the highest point. Upon termination of processing the centerboard on both side frames, we try to get a result with higher confidence by combining both results. The process of selecting the centerboard result is depicted in algorithm 5.6. First, obviously, if one of the results is empty, we select the other non empty result. If the results are close, an average is returned. Otherwise, we try to make a decision based on a higher confidence value. In case of equality, if the confidence is low, there is a rather high chance the detected “centerboard” is a mistake. Thus, in this case, if the confidence is low, we select the minimal result, while when it is high, we select the maximal result.

5.2.6 Wind Indicator Detection

There are several Android applications available on Google Play that attempt to serve as anemometer and to measure the wind angle and speed. Some of them extract the wind speed by analyzing the spectrum of sound. We tried to use and integrate one of these applications into our sailing trainer. However, the given results were not sufficiently accurate. An additional option used by anemometer applications available on Google Play is to present information about the wind acquired by a nearby weather station. However, we
Algorithm 5.5 Centerboard Detection on a single side frame pseudo code

{Input: \( Img \) image from the side camera}
{Output: Centerboard height and confidence in the result}

1: \( \text{cropped} := \text{crop}(Img); \)
2: \( \text{cropped} := \text{smooth}(\text{cropped}); \)
3: \( \text{cropped} := \text{matrixOperations}(\text{cropped}); \)
4: \( \text{contours} := \text{findContours}(\text{cropped}); \)
5: if (empty(contours)) then
6: \( \text{return} 0; \)
7: end if
8: \( \text{areaSum} := 0; \)
9: \( \text{maxLength} := 0; \)
10: \( \text{highestContour}, \text{secondHighestContour} := \text{null}; \)
11: \( \text{maxLength} := 0; \)
12: \( \text{minX}, \text{minY} := \text{INF}, \text{maxX}, \text{maxY} := 0; \)
13: for each contour in contours do
14: \( \text{maxLength} := \max(\text{maxLength}, \text{contour.height})(); \)
15: if (canEliminate(contour)) then
16: \( \text{continue}(\text{contour}); \)
17: end if
18: \( \text{areaSum} := \text{areaSum} + \text{area}(\text{contour}); \)
19: \( \text{update}(\text{highestContour}, \text{secondHighestContour}); \)
20: \( \text{update}(\text{minX}, \text{minY}, \text{maxX}, \text{maxY}); \)
21: end for
22: \( \text{centerboardLength} := \text{maxY} - \text{minY}; \)
23: \( \text{distance} = \text{distance}(\text{highestContour}, \text{secondHighestContour}); \)
24: if (distance > threshold) then
25: \( \text{centerboardLength} := \text{centerboardLength} - \text{distance}; \)
26: end if
27: \( \text{confidence} := \text{howConfident}(\text{minX}, \text{minY}, \text{maxX}, \text{maxY}, \text{areaSum}); \)
28: if (areaSum < centerboardLength * constant) then
29: \( \text{centerboardLength} := \text{maxLength}; \)
30: end if
31: \( \text{return} (\text{centerboardLength}, \text{confidence}); \)

Figure 5.7: Centerboard processing
Algorithm 5.6 Combining Centerboard length calculation from both side frames to determine its height
{Input: The tuples \((\text{centerboardLength}_1, \text{confidence}_1), (\text{centerboardLength}_2, \text{confidence}_2)\) which contain the centerboard length and the confidence in the measurement as deduced from both side frames} {Output: Centerboard height}

1: \(\text{if} \ (\text{empty}(\text{centerboardLength}_1) \ \text{OR} \ \text{empty}(\text{centerboardLength}_2)) \ \text{then}\)
2: \hspace{1em} \text{return nonempty}(\text{centerboardLength}_1, \text{centerboardLength}_2); \quad \text{end if}\n3: \hspace{1em} \text{if} \ (\text{abs}(\text{centerboardLength}_1 - \text{centerboardLength}_2) < \text{threshold}) \ \text{then}\n4: \hspace{2em} \text{return average}(\text{centerboardLength}_1, \text{centerboardLength}_2); \quad \text{end if}\n5: \hspace{1em} \text{if} \ (\text{confidence}_1 \neq \text{confidence}_2) \ \text{then}\n6: \hspace{2em} \text{return maxConfidence}(\text{centerboardLength}_1, \text{centerboardLength}_2); \quad \text{end if}\n7: \hspace{1em} \text{if} \ \text{confidence}_1 < \text{threshold} \ \text{then}\n8: \hspace{2em} \text{return min}(\text{centerboardLength}_1, \text{centerboardLength}_2) \quad \text{end if}\n9: \hspace{1em} \text{return max}(\text{centerboardLength}_1, \text{centerboardLength}_2) \quad \text{end if}\n
Figure 5.8: Example images, before zooming where the wind indicator is marked by a green rectangle

need more accurate results. Eventually, due to the above reasons we use the camera to analyze the wind indicator angle and deduce the wind direction relative to the boat, but omit the wind speed.

As noted, we are detecting the angle of the wind indicator in order to deduce the wind angle. The wind indicator is attached to a rod. This rod is a continuation of the boom. Thus, the angle we are looking for is the angle between the wind indicator and the rod it is attached to. A red cylinder is attached to the rod and has the same slope as the rod. This means that we can determine the slope of the rod by determining the slope of the red cylinder. The wind indicator on the original image (prior to any processing) is depicted in Figure 5.8. We can use both side cameras, as both capture the wind indicator. Nevertheless, we use only the camera which is on the same side as the sail is. We use only this camera as in the pictures taken from this camera the wind indicator is mostly on the sky background and thus more easily detected. Therefore, we should first decide on which side of the sailboat the boom is located.

As the camera is installed on the sailboat, we can know approximately where the wind indicator is. The first step of our algorithm is zooming on the wind indicator. As noted previously, it enables us to eliminate distortions and to speed up the processing time. The resulting images can be seen in Figure
5.9. One can notice that the color of the object we need to detect is rather unique. Therefore, we defined a color range to be large enough to include the wind indicator hue range (in different light conditions), while being as small as possible to eliminate other elements which are close in color. We applied the transformation by subtracting and checking the matrices of the relevant colors. The resulting pictures after transformation can be seen in Figure 5.10.

At this point, we tried to detect contours of relevant size and characteristics. We first iterated over all the contours, while skipping contours which are not probable to represent neither the wind indicator nor the red markers on the rod. For instance, if the contour is bounded in a very narrow rectangle and is located in the bottom of the frame, or if its bounding rectangle size is smaller or larger than a predefined size or if the bounding rectangle is mostly empty (the contour area is very small). While going over the contours (and before skipping any of them) we keep track of the largest candidate in terms of largest binding rectangle, we will refer to it later as the largest candidate.

In addition, while iterating over the contours we tried to detect the rectangle containing the wind indicator and the red cylinder on the rod. We did it by applying heuristics about the location, size and the percentage of the counter area from its bounding rectangle.

Sometimes we had only one suitable contour. This happened usually when the wind indicator pointed to the rod and the red markers on the rod were merged to the same contour as the wind indicator. In addition, if no suitable contours were found or only a single small contour was selected, we took the largest candidate as a single selected contour. The last case in which we would have a single candidate is when the two found rectangles were separated incorrectly. In this case we combine both rectangles to a single rectangle containing both. We conclude that such mistake happened by observing the gap between both candidates. We estimate what the slope of the wind indicator is, given that it crosses both candidates
and, afterwards, try to check if it makes sense given the two candidates. See example in Figure 5.11.

In the above cases (when we ended up with a single rectangle), we tried to detect two intersecting lines and decide which line represents the wind indicator and which line represents the rod the wind indicator is connected to. First we took the bounding rectangle of the contour. If it was too small, we enlarged it to some predefined constants. In order to detect lines on the resulting rectangle, we used Canny edge detection followed by Hough Transform for lines algorithm. We applied these algorithms on the image without any color transformations as the rod is mostly colored in a non unique color. We then tried to detect the wind indicator line. The selected line’s edge should be close to the corners of the bounding rectangle (as usually it defines the contour’s edges), and its slope should be rather close to the rectangle diagonal (for the same reason). The slope should be rather close to the rectangle diagonal and not identical, as the wind indicator has a triangle in its end (and it is not merely a straight rod). The characteristics explained above guided us when selecting the line representing the wind indicator. Afterwards, we selected the line representing the rod, which should be closer to the sail and check its intersection point with the wind indicator. The rod line should not be a continuation of the wind indicator line (thus, we tried to detect and eliminate such lines). In some cases we could not detect any lines (resulting in an empty result). In order to overcome these cases, we enlarged the area in which we are looking for the lines (handling cases in which the lines were cut off), and applied Hough Transform for lines again (with a lower threshold). The resulting picture can be seen in Figure 5.12. The green rectangle is the bounding rectangle of the selected contour, while the white lines are the lines that represent the deduced angle.

In some cases, we got more than one contour. If we had more than 2 contours, we first decided what contours are relevant. We went over each contour and checked how probable it is to be the wind indicator contour and for each other contour how probable it is to be the red cylinder contour for this wind indicator contour. We took into account multiple aspects of the candidate pair location relative to one another and their sizes.

After selecting two rectangles, we applied the Fitellipse OpenCV algorithm [BK08] to the contour containing the red cylinder. The slope of the resulting ellipse in most of the cases represents well the slope of the rod.

The second contour should contain the wind indicator. Therefore, we tried to detect a line representing the wind indicator slope. First we tried to apply Canny edge detection algorithm followed by Hough
Transform for lines algorithm. If this did not yield any lines, as the wind indicator is red, we first applied matrix operations in order to identify its unique color, followed by Canny edge detection and found lines by Hough Transform for lines. Next, we selected the line to represent the wind indicator. We checked the intersection points of the lines with the picture edges. In addition, if the slope was very close to the slope deduced for the rod, and the location was rather suitable for the rod, we ignored the line. An additional factor was how many lines support the same slope (as often several lines were found over the wind indicator). When we had found the wind indicator slope we were not yet done, as we should also know in which direction it points. The wind indicator has a triangle in one of its edges and we should decide in which direction it points.

One technique we have considered was sampling random points in the corners of the bounding rectangle of the wind indicator and selecting the corner which had the largest amount of points. Unfortunately, this technique did not perform well, as sometimes, due to the wind indicator angle, the triangle seemed rather narrow and the selected corner was not always correct. The next approach we had considered was to check the distance between points we have set on the bounding rectangle borders to the contour.

Mostly, we tried to detect the corner to which the wind indicator points after detecting the line that represented the wind indicator. In this case, we checked only two corners. However, sometimes we could not find a suitable line, and thus we checked the four corners. We will explain shortly how we found the wind indicator slope in this case. This approach performed well except for when the wind indicator is vertical. In this case, we checked the distances a bit differently, and selected the bigger distance.

Sometimes, no lines are found on the selected rectangle to represent the wind indicator. In this case we first tried to deduce the slope by the bounding rectangle diagonal. This did not give sufficient results. As we noted before, due to the triangle at the wind indicator end, the diagonal does not always represent the wind indicator slope. Therefore, according to the corner to which the wind indicator points, we went over the sides of the rectangle and measured the distance from the contour. Upon reaching a zero distance or the smallest distance, we decided about the slope. Some examples of correct detections can be seen in Figure 5.13.

We also tried to decide about the direction of the wind indicator by the distance between the red cylinder and the wind indicator. However, it did not work well, as it is very hard to employ a correct threshold.
We note that the wind indicator detection process described above does not take into account the distortion in the wind indicator angle that is caused by the location of the camera relative to it. A linear transformation should be applied to the result of this process in order to deduce the correct angle of the wind indicator relative to the rod.

5.2.7 Telltale Detection

We detect the telltale after the boat’s centerline angle and the boom angle relative to it were calculated. Thus, we know to which extent and to which side relative to the boat’s center the sail is opened. The side of the sail enables us to determine which camera to use (the telltale that interests us is the one on the same side as the sail is, i.e., the outer or down-wind telltale), while the angle determines the approximate location of the telltale relative to the boat. Therefore, the first thing we do is to take the picture from the relevant camera and crop it according to the angle of the sail.

Previously, we tried to crop the relevant area after detecting the sail by looking for three lines that intersect and form a triangle (which represents the sail). But this was error prone and required a rather long time to apply manipulations on the picture and select the correct lines. Figure 5.14a is an example of this zooming attempt. The red lines are all the lines found by the Hough Transform lines algorithm, while the selected lines are marked in green.

We also made attempts to reduce our search area by locating the digits and letters placed on the sail. In order to do that we employed OCR (Optical Character Recognition). But probably, as the text is not vertical, the implementations we tried have not detected the letters. We have not tried to use Deskew algorithm for the OCR as we just deduced that we can zoom in a sufficient extent by knowing the sail angle relative to the boat center.

Ultimately we arrived at the following solution. We duplicated the image. In one instance of the image, we tried to find the telltale base candidates (this image is denoted as base image) while in the other, we tried to detect the telltale candidates (this image is denoted as telltale image). On the base image we applied an erosion morphological operator in order to eliminate noise and protrusions and enlarge the telltale’s base [BK08]. Following that operation, we applied Gaussian smoothing of the image. Previously, we tried to detect the telltale base by matching a predefined template, but this did not work well, probably as the sail moves and the structure and the size of the telltale base changes as well as the lightening on the picture. The template we applied is shown in Figure 5.14b. Currently, we do not use it. The telltale
base in this template is blue as in the first sessions we have worked on, the telltale base was blue (and not
green or red).

Eventually, in the sessions we present in this work, on one side of the sail both the telltale and the
telltale base are colored in red while on the other side they are green. Each side picture was passed to the
telltale detector class with a class that exposes a function that identifies its specific color according to side
of the sail it is located on. Thus, we employed the function relevant to each side.

On pictures where the telltale base was blue while the telltale itself was red, we performed the
following: We created two matrices from the base image. One containing the red channel value of every
pixel of the base image while the other, contains the blue channel value of each pixel. We then subtract
the latter matrix from the former. We did this operation as in the telltale base, blue channel value is greater
than the red channel value and on the sail the opposite holds. Before that, we tried to identify the exact
color of the base and to apply transformations in order to identify it. However, it gave results which were
very close to the channels subtraction and was much more complex. On the telltale image we applied
matrix operations to identify regions that have the same color range as the telltales.

Currently, we perform erosion on the base image followed by a Gaussian Blur and matrix operations
to identify the concrete color. We then find contours on the base image. If the area of the contour does
not fit telltale base size, we eliminate it. We disregard also regions on the edge of the cropped picture (as
we took our cropping range to include the base and a radius around it for searching for the telltale). As
we know, the contour should be located on a sail which has a specific color. Thus, we draw a rectangle
around the base candidate and count the amount of dots that have the sail’s color and are located on this
rectangle. If this number is lower than some threshold, we ignore this contour.

We also check if the contour found contains a circle of the base color by or around it. For that we
build a hash map of points and the “grade” they got in terms of how probable they are to be considered
the base center. Therefore, we check for each point if its color fits the base. If it is, we enlarge the grade
to every point \((x_0, y_0)\) in a predefined radius \(r\) from \((x, y)\) and for every point \((x_1, y_1)\) in a predefined
radius \(r/2\) from \((x, y)\).

Afterwards, we check the surrounding of each candidate that got a grade higher than some threshold.
For every point \((x_0, y_0)\) in a predefined radius \(r_1\) from the candidate point \((x, y)\) if its color is the telltale
base color, we lower the grade (until at some point we just eliminate the candidate), while if it fits the
surrounding sail color we enlarge its grade. As the surrounding calculation is done separately for each
candidate and does not depend on the other candidates, we made this calculation run in parallel, which resulted in execution time improvement. Next, we take a predefined amount of top candidates (which have the highest grades) while sorting them by grade. Finally, we merge the results by their location.

At this point, we have several telltale base candidates and we try to find the telltale itself. We take the telltale image and mark edges on the H channel element (out of the HSV values) by Canny edge detection and look for straight lines with the help of Hough Transform for lines algorithm. For each of the detected lines we go over all the possible bases and if the line is close enough to the base candidate and is positioned in a possible manner relative to it, we associate it with the base candidate. At this point, we try to merge bases even more and combine their lines. If the bases are located horizontally, their grades are drastically decreased (this is not probable), while if they are located vertically the grades are increased (as in this way the telltale from the other side of the sail is seen). We try to filter more lines by:

- Location on the sail - if we draw parallel lines from both of its sides and we see that it is surrounded by relevant color.
- Distance from the telltale base center
- Being a part of a lot of intersecting lines.

If after this filtering we have a base with no telltale lines surrounding it, we dismiss it. Next, we go over all the lines and grade them based on the amount of "sail" points that surround it, location and angle relative to the telltale base and the grade that got the telltale base. In addition, if the line is short, we also take into account if the line is continued to some direction (not necessarily by a straight line) while if it is long, we lower its grade. Eventually, a single line which has the maximum grade is selected. If a few lines have the same grade, the last line is taken (arbitrary choice). As we use Hough Transform for lines algorithm to find the telltale, if the whole telltale has no straight sections it will not be detected.

Another consideration that one should not forget is that the telltale angle should be measured relative to the boom and not to the horizon. We noticed that when assuming constant location of the cameras, the boom slope relative to the horizon depends on the extent the sail is opened. Thus, given the angle of the sail relative to the boat center, we calculate the telltale slope relative to the boom. We could achieve a more accurate measurement if we would place a line on the sail and measure the telltale slope relative to this line. However, eventually we decided to skip this approach.

In addition, the telltale can be a straight line with a correct slope, pointing from its base, but to the opposite side. The described situation is undesirable. We conclude that the telltale points to the other way around by its location on the x axis relative to its base and its length.

5.2.8 Image Analysis Fine Tuning

In all the image analysis algorithms, we make a lot of fine tuning and adjusting of the parameters in order to get better results. When we use Hough Line Transform, we should define the length of the line we are looking for, the maximum gap between two points to be considered in the same line, and a minimum number of intersections to detect a line. As Hough Transform uses intersections between curves to form a line, this is a actually a threshold. When we perform matrix operations to identify some color range by
Algorithm 5.7 Telltale detection on a single side frame pseudo code

INPUT: Img image from the relevant side camera, sailAngle by which the sail is opened and to which side - sailSide

OUTPUT: Telltale angle relative to the boom

1: cropped := crop(Img, angle);
2: telltaleImg := clone(cropped);
3: cropped := erosion(cropped);
4: cropped := smooth(cropped);
5: cropped := matrixOperations(cropped);
6: contours := findContours(cropped);
7: if (empty(contour)) then
8:     return null
9: end if
10: for each contour in contours do
11:     if (canEliminate(contour)) then
12:         continue;
13:     end if
14:     for each point in contours do
15:         if (fitsBase(point)) then
16:             pointToGrade := addGrade(centerRelativeToPoint);
17:         end if
18:     end for
19: end for
20: for each point in pointToGrade do
21:     pointToGrade := gradeBySurrounding(point);
22: end for
23: baseToGrade := findMaximum(pointToGrade);
24: mergeResults(pointToGrade);
25: boostCloseBases(pointToGrade);
26: hueChannel := hueChannel(telltaleImg);
27: hueChannel := Canny(hueChannel);
28: lines := HoughLinesP(hueChannel);
29: baseTelltale := fill(baseToGrade, lines);
30: baseTelltale := filterAndMergeAgain(baseToGrade, baseTelltale);
31: removeEmptyBases(telltaleToLines);
32: winningLine := gradeLines(telltaleToLines);
33: slope := calcSlope(winningLine, sailAngle, sailSide);
34: if (shouldSwap(winningLine, winningBase)) then
35:     slope := 180 − slope;
36: end if
37: return slope
comparing the color channels to some predefined values and comparing them one to the other, there is also a lot of fine tuning. The smoothing, dilation and erosion algorithms all have parameters to be tuned and our heuristics are tuned to achieve better results. In this section we analyze the impact of some of these decisions.

In Figures 5.14 and 5.15 we check the impact of varying the line length we are searching in the Hough Transform for lines algorithm and the maximum gap between intersections that can be considered as a single line. We took a small sample of 50 images and we can see that even small changes in those parameters yield differences in the results.

![Figure 5.14: Tiller angle detection accuracy percent in varying line lengths parameter of Hough Transform for lines](image)

![Figure 5.15: Tiller angle detection accuracy percent in varying max line gap parameter of Hough Transform for lines](image)
5.2.9 Image Analysis Light and Weather Conditions

The light and weather conditions of the images our program gets as input vary highly. This mostly effects the pictures taken by the side cameras. Figures 5.16 and 5.17 compare the centerboard and the telltales as they are seen in different lightening conditions. In case the lightening is dimmed over the telltales, it is frequently hard to distinguish between the telltale and the underlying sail.

Due to these differences, we created three side light categories and two upper light categories. We determine the lightening level for each session and following this decision, the Image analysis is changed according to the lightening.

The only difference in the case of the upper light is in the tiller detection as, in this case the surface of the boat is darker and thus our algorithm needs to be adjusted (as colors that used to represent the tiller on lighter images can represent the surface of the boat on the darker images). An example of the difference can be seen in Figure 5.18.
5.2.10 Image Analysis Evaluation

In order to evaluate the correctness of the image analysis, we had to go over each of the images manually and record the correct results for all the image analysis aspects for it. In order to make the process easier (as we went over hundreds of pictures), we iterated over the results from our image analysis and corrected the wrong results.

We did not measure the angles manually but looked if the angle looks close enough, as measuring the angles manually on each picture exactly consumes much time. Moreover, in order to decide about the instructions, we need a close estimation of the angle, and not the exact angle. Thus, also when we check the correctness, we check that the angle is in a range of a few degrees distance from the correct result.

In the sailor’s detection, if the detected point was upon the sailor shirt, mostly we classified it as a correct result, as it is very challenging to estimate exactly where the sailor sits (we do not know if the sailor leans and if he does, does he lean backwards or forwards). Therefore, we try to estimate where the center of his mass is located. We say that “mostly we classified it as a correct result”, as in cases it caused a definite error we have regarded it as an error. In addition, the sailor detection correctness estimation estimates both his location inside the cockpit and the side of the boat to which he sits.

In the cases of the centerboard and the telltale, eventually we have split the results to classes when each class represents a set of results. Three classes were constructed for the centerboard and two classes for the telltale results. The percentage of the correct results, rounded to the closest integer, are displayed in Table 5.1.

In sailing session #1, we did not evaluate the centerboard as in this session a compass was installed in front of the centerboard and concealed a major part of it. In sailing session #4, we have not measured the wind indicator as it was broken and fixed with white isolation band which does not fit our image analysis assumptions by which the wind indicator is red.

As can be seen from the table, the image analysis tasks of the upper frame generally gave better results than the side processing task for several reasons. First, there are more lightening issues on the side frames. In addition, the image analysis tasks in the side frames were generally more challenging. For example, a major challenge in the image analysis phase of detecting the centerboard is that the legs of the sailor are located exactly in the same area in some cases and have a very similar color.
<table>
<thead>
<tr>
<th>Component</th>
<th>session 1</th>
<th>session 2</th>
<th>session 3</th>
<th>session 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of images</td>
<td>415</td>
<td>603</td>
<td>808</td>
<td>1305</td>
</tr>
<tr>
<td>Tiller angle</td>
<td>89</td>
<td>89</td>
<td>92</td>
<td>93</td>
</tr>
<tr>
<td>Tiller side (left right)</td>
<td>97</td>
<td>93</td>
<td>94</td>
<td>94</td>
</tr>
<tr>
<td>Sail angle</td>
<td>92</td>
<td>92</td>
<td>93</td>
<td>95</td>
</tr>
<tr>
<td>Sailor location</td>
<td>98</td>
<td>98</td>
<td>99</td>
<td>95</td>
</tr>
<tr>
<td>Centerboard position</td>
<td>-</td>
<td>67</td>
<td>85</td>
<td>55</td>
</tr>
<tr>
<td>Telltale angle</td>
<td>82</td>
<td>81</td>
<td>63</td>
<td>72</td>
</tr>
<tr>
<td>Wind indicator angle</td>
<td>61</td>
<td>67</td>
<td>48</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.1: The percentage of the correct results (rounded to the closest integer) of the different components in different sessions
Chapter 6

Decision Process

6.1 Heuristics Based Paradigm

Here, the autonomous sailing trainer responds to the inputs according to a set of predefined rules. The rules for the heuristic are based on Gil Manor’s advise in addition to responses collected while observing regular training sessions. Figure 6.1 represents the initial heuristic flow we had in the system.

After employing an initial heuristic flow, we have analyzed the errors that occurred. Following this analysis, we have changed and refined the heuristic. In addition to adding additional parameters that were checked (for instance, the centerboard) while removing others, such as wind speed which eventually, we could not measure. We also check different patterns in data, for instance we check if the tiller position...
“jumps” or changes rapidly in order to warn the sailor that he should use the tiller less. We take into account different sort of averages, for instance we calculate average while ignoring instant picks, give more weight to the last value or check if some readings (for instance the compass) are rather constant. In addition, as explained in section 9.1, we often look in two adjacent frames to avoid singular errors and also give a chance to the sailor to correct a mistake. All these changes made the heuristic module much more complicated than the one presented in figure 6.1. In figure 6.2 we illustrate the heuristic we currently have, without the specific values and the concrete conditions. It only shows which parameter refers to which output and the manipulations done over the parameters are specified on the the arrows. The concrete values and conditions are specified in table 6.1. An additional aspect that is not represented in this figure is that if the boom has changed sides in one of a predefined amount of last frames, we return an empty output. As in the case of tacking we are more lenient to mistakes.

6.2 Machine Learning Paradigm

We now report on our machine learning experience. Let us note that each image might correspond to more than a single instruction. For instance, we might suggest to take the centerboard out and to flatten the boat. Thus, we attempt to design a multi-label classifier, where multiple labels might be attached to the same data instance.

As noted in [RPHF09], splitting the multi-label classifier into one or more single-label classifiers is a common approach to the multi-label classifying problem. This approach has additional advantage that it enables us to use the machine learning paradigm for some components for which it proved to give better results while employing the heuristics paradigm to others (see further discussion in Section 6.3). Hence, we had split the multi-label classifier into several classifiers, which allowed us to specify a single class attribute in each.

As described in section 5.2.10, the image analysis results contain errors. Therefore, a question is raised as to whether to train and test the classifier with correct or erroneous data. We evaluated three different models. One was trained and tested on correct input (denoted as ModelA), the second was trained and tested on erroneous data (denoted as ModelB) and the third was trained on correct data, but tested on erroneous data (denoted as ModelC).

We created six single-label classifiers:

1. Centerboards related instructions classifier
2. Sail related instructions classifier
3. Tiller related instructions classifier
4. Sailor location related instructions classifier
5. Boat flattening related instructions classifier
6. Ropes related instructions classifier
<table>
<thead>
<tr>
<th>Command</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take center board out</td>
<td>current sail angle &lt;100 AND previous sail angle &gt;35 AND centerboard in the last two frames &lt;20</td>
</tr>
<tr>
<td>Put center board in</td>
<td>current sail angle &lt;30 AND centerboard in the last frame &gt;100 AND centerboard in the previous frame &gt;20</td>
</tr>
<tr>
<td>Put center board in</td>
<td>current sail angle &lt;30 AND centerboard in the last frame &gt;30 AND centerboard in the previous frame &gt;30</td>
</tr>
<tr>
<td>Put center board in</td>
<td>current sail angle &lt;30 AND centerboard in the last frame &gt;20 AND centerboard in the previous frame &gt;100</td>
</tr>
<tr>
<td>Move to front</td>
<td>previous sailor position &gt;60% of the cockpit AND current sailor position &gt;65% of the cockpit</td>
</tr>
<tr>
<td>Move to front</td>
<td>previous and current sailor position &gt;55% of the cockpit AND current sail angle &lt;30</td>
</tr>
<tr>
<td>Loose the sail</td>
<td>Compass is constant AND telltale angle in the last two frames &gt;28 AND current sail angle &lt;60</td>
</tr>
<tr>
<td>Loose the sail</td>
<td>Compass is constant AND tiller and sailor are on the same side AND tiller average angle &gt;15 AND current sail angle &lt;60</td>
</tr>
<tr>
<td>Loose the sail</td>
<td>Compass is constant AND telltale angle in the last two frames &gt;28 AND current sail angle &lt;60</td>
</tr>
<tr>
<td>Tight the sail</td>
<td>Compass is constant AND Average wind angle (ignoring instant picks) &lt;10 AND sail last angle &gt;20</td>
</tr>
<tr>
<td>Push the tiller</td>
<td>Compass is constant AND telltale angle in the last two frames &gt;28 AND current sail angle &lt;45</td>
</tr>
<tr>
<td>Ease the ropes</td>
<td>Boom angle average in 3 frames &lt;35 AND boom angle average in the following 2 frames (last frames) &gt;50</td>
</tr>
<tr>
<td>Ease the ropes</td>
<td>boom angle average in two last frames &gt;50 AND Ease the ropes was said in the previous frame (due to the above condition)</td>
</tr>
<tr>
<td>Use the tiller less</td>
<td>In the last 5 frames there are at least 3 large changes in the tiller (large change is over 16 degrees in its angle, changes are accumulated over subsequent frames).</td>
</tr>
<tr>
<td>Flatten the boat</td>
<td>Compass is constant AND current sail angle &lt;60 AND last orientation &gt;10 AND orientation average - double weight to the last two values &gt;20</td>
</tr>
<tr>
<td>Flatten the boat</td>
<td>Compass is constant AND current sail angle &lt;60 AND orientation average - double weight to the last two values &gt;15 AND orientation is not towards the sailor</td>
</tr>
<tr>
<td>Flatten the boat</td>
<td>current sail side is right AND current sail angle &gt;60 AND orientation average - double weight to the last two values &gt;14 AND ((compass is decreasing AND negative orientation of the boat) OR (compass is increasing AND positive orientation of the boat))</td>
</tr>
<tr>
<td>Flatten the boat</td>
<td>current sail side is left AND current sail angle &gt;60 AND orientation average - double weight to the last two values &gt;14 AND ((compass is increasing AND negative orientation of the boat) OR (compass is decreasing AND positive orientation of the boat))</td>
</tr>
<tr>
<td>Flatten the boat</td>
<td>current sail angle is towards the sailor AND orientation average - double weight to the last two values &gt;15 AND last orientation &gt;10</td>
</tr>
<tr>
<td>Flatten the boat</td>
<td>Orientation maximal value - orientation minimal value &gt;= 15 AND there are at least 2 large changes in the boat orientation - 7 is set as a threshold</td>
</tr>
<tr>
<td>Flatten the boat</td>
<td>Compass is constant AND (orientation average - double weight to the last two values &gt;12 AND orientation last value &gt;9) OR (orientation average - double weight to the last two values &gt;15))</td>
</tr>
</tbody>
</table>

Table 6.1: Commands and the concrete heuristic
For each classifier, we took the input from the sensors for the given images and Gil’s instruction as an output. The files were fed into Weka [Wek]. For models A and B, we implied cross validation with 10 folds to evaluate the classifier’s accuracy. While for model C we performed a cross validation, the test data was taken from the erroneous data, while the training set was taken from the correct data. We partitioned the pictures into 10 random equal parts. For each of the 10 parts we created a test file with the corresponding erroneous part, while for the complementary 9 parts we created a training file, containing the correct data. We took the average of the 10 results.

The input file for Weka consists of the following fields defined in the .arff file:

- boatOrientation[NUMERIC] - the roll angle of the boat - we feed 5 last values.
- compass[NUMERIC] - the compass reading.
- boatSpeed[NUMERIC] - the speed of the boat.
- prevSailorPosition[NUMERIC] - sailor’s position in the cockpit in the previous frame.
- sailorPosition[NUMERIC] - sailor’s position in the cockpit.
- sailorSide[L,R] - on which side of the boat the sailor sits.
- prevBoomAngle[NUMERIC] - the angle of the boom relatively to the boat in the previous frame.
- boomAngle[NUMERIC] - the angle of the boom relatively to the boat.
- prevBoomSide[L,R] - to which side of the boat the boom is opened in the previous frame.
- boomSide[L,R] - to which side of the boat the boom is opened.
- prevCenterBoard[0,1,2] - the extent by which the centerboard is seen above the boat in the previous frame (0 = not seen, 1 = half seen, 2 = fully seen).
- centerBoard[0,1,2] - the extent by which the centerboard is seen above the boat (0,1,2 meaning are as of the item above).
- prevtelltale[0,1] - the position of the telltale relative to the boom in the previous frame (0 = correct = telltale angle relative to the boom $\leq$ 28 degrees, 1 = wrong = telltale angle relative to the boom $>$ 28 degrees).
- telltale[0,1] - the position of the telltale relative to the boon (0,1 meaning are as of the item above).

- Wind fields
  - avgWindAngle[NUMERIC] - the average wind angle over the frames.
  - flatAvgWindAngle[NUMERIC] - the average wind angle over the frames ignoring one time glimpses.
  - maxWindAngle[NUMERIC] - the maximal wind angle over the frames.
  - minWindAngle[NUMERIC] - the minimal wind angle over the frames.
• Tiller fields
  – tillerSide[L,R] - on which side of the boat the tiller is located.
  – avgTillerAngle[NUMERIC] - the average tiller angle over the frames.
  – flatAvgTillerAngle[NUMERIC] - the average tiller angle over the frames ignoring one time glimpses.
  – maxTillerAngle[NUMERIC] - the maximal tiller angle over the frames.
  – minTillerAngle[NUMERIC] - the minimal tiller angle over the frames.

• the output instruction that should be given to the sailor (a single output for each of the 6 classifiers, as previously described):
  – action[center-board-out,center-board-in,none]
  – action[loose-sail,tight-sail,none]
  – action[push-tiller,pull-tiller,less-tiller,none]
  – action[move-to-front,none]
  – action[flatten-boat,none]
  – action[ease-ropes,tight-ropes,none]

We chose to use Weka’s J48 classifier which represents the C4.5 decision tree. We selected this model as it is able to deal with numeric attributes, noise in the data and missing values as described in [WFH11]. It is also noted in this book that the C4.5 system is the Machine Learning system most used nowadays. In addition, we can easily see and understand the created tree and it can be easily split or modified if needed, for instance, if we want to replace some part of the decision tree with the heuristics we have created.

6.3 Evaluation and Comparison

We have created a session of pictures and logged the reading of the other sensors during this session. Afterwards, we have created pictures that summarize every two seconds of the sailboat and sailor conditions in the last 10 seconds, as shown in Figure 6.3. Gil Manor, the coach of the Laser team in the Haifa Sailing club as well as the national team, has gone over these pictures and reviewed the proposed output of the system. By Gil’s remarks we constructed a “Golden Truth” to which the output of the system will be compared and by comparing to these results it will be evaluated. Plot 6.4 summarizes the percent, rounded to the closest integer, of correctly classified instances with the heuristics versus the machine learning mechanism. We classified and compared 2 sessions: numbers #2 and #3 that were referred to in section 5.2.10.

As we have already mentioned, we split the multi-label classifier to multiple single label classifiers as this allows us modularity in terms of selecting the most appropriate classifier for every component. In addition, for some components we cannot employ the machine learning paradigm as we do not have enough classified example data. This is the case in the ropes classifying. The 100% correctness is misleading there as it results from the minor amount of those instances in the classified data.
The 6 classified instances in plot 6.4 correspond to the 6 single labeled classifiers presented in 6.2. The x-axis represents the following classifiers:

1. Heuristics paradigm: we employ the predefined heuristics we have created to specify the output for each set of input parameters. Corresponds to section 6.1.

2. Machine learning - correct data: we classified and performed cross validation on the instances while getting the manually corrected input. Corresponds to Model A in Section 6.2.

3. Machine learning - erroneous data: we classified and performed cross validation on the instances while getting the erroneous data as input. Corresponds to Model B in Section 6.2.

4. Machine learning - combined: we classified the instances while getting the correct data as input but performing testing on the erroneous data. Corresponds to Model C in Section 6.2.

We can also deduce from the measurements that learning on the correct data has not yielded better results than the same process with erroneous data. Thus, learning upon the erroneous data is preferable. The second column in Figure 6.4 evaluates the machine learning upon perfectly classified data. In other words, we evaluate a case in which we could reach almost 100% correctness in acquiring the data regarding boat’s and sailor’s condition. This measurement indeed mostly shows an improvement when comparing to the learning process upon erroneous data.

It is important to note that while correcting the image analysis results in the data, we did not make any changes in the readings of the phone sensors. We did not make any changes, as first, we assume that the sensors are rather accurate and second, if we would have corrected this data we would rely on a single image that represents a single frame of time once in two seconds, which is not necessarily more accurate than the sensors’ readings.

We have also checked the precision and recall of the different commands in plots: 6.5a, 6.5b, 6.5c, 6.5d and 6.5e. In these plots precision is denoted as P and recall is denoted as R. It turned out that we were not able to accumulate enough errors in the sessions we have recorded. Thus, the precision and recall are worse than the accuracy. Nevertheless the ability of the autonomous not to say instructions when they should not be said is also significant.
Figure 6.2: Current heuristic decision flow
Figure 6.3: An example picture that was presented to a sailing coach in order to construct a “Golden Truth”
Figure 6.4: Evaluation of the machine learning versus the heuristics paradigms in terms of the amount of correctly classified instances.
Figure 6.5: Precision and recall of the different commands in different classifiers
Chapter 7

Application Life Cycle

Upon the initialization of the application, no sensors are activated. Only when the sailor presses the start button all the sensors are initialized and started. Velocity and Orientation sensors are notified about changes in their readings and store the results locally, while the Image analysis sensor activates the camera in the correct mode and start sampling the pictures. Currently, it takes too long to download the pictures, as we have already explained. Thus, we have tested the different parts of the application, but not all the working system as a whole. Upon receiving pictures from the cameras, they are stored in a circular FIFO queue and the queue lock is notified. The queue enables us to throw away outdated pictures when the queue is overfilled. A PictureProcessingThread is also activated by the Image analysis sensor on start up. If the queue is empty, this thread waits upon the pictures queue lock and once it is notified, it takes the first stored pictures and processes them. Once the pictures are processed, it alerts the Main activity (SailingHelperActivity) which activates a worker thread ProcessMeasurements to handle the alert. This thread queries the other sensors and constructs all the known input information about the current condition. Afterwards, this information is given to a strategy class to decide about the algorithm’s output (if the sailor helper should suggest something to the sailor, and what should he suggest). The output is given to a Speaker class which says the instructions aloud.

The sailor can stop the application by pressing the stop button. When this button is pressed, all the sensors are disabled, the listeners are unregistered and the cameras are turned off. A high level sequence diagram of the sailing helper activation can be seen in Figure 7.1.

7.1 Connecting the Cameras

The GoPro HERO3 White Edition cameras have a Wi-Fi built in. Each camera creates a Wi-Fi network to which one can connect in order to activate the camera in the correct mode and download the taken images.

The cameras do not expose an official API for connecting and interacting with the camera from within another application, thus we have used HTTP requests described in GoPro Java API project [gop] to interact with the camera. Upon starting the application, we turn on the cameras, set it to the correct mode and start capturing pictures. These commands of interaction with the camera are sent from a worker thread, to keep the UI thread responsive.

We initiate a command queue for the camera related tasks. A worker thread processes the requests
one by one. In case the request did not return an HTTP OK 200 code, the thread waits for a predefined amount of time and then retries. In addition, the thread waits for a predefined amount of time between the commands. This thread is terminated when all the required commands were executed.

The cameras are configured to take a picture each 2 seconds as this was considered a reasonable frequency by a sailing coach. The cameras are started in the same parity of seconds in order to keep all the pictures synchronized. For instance, in one of the sessions the first camera was started on 1326734138021, the second camera 1326734154017 and the last one on 1326734170010. Thus, the maximal split between the three cameras in the described case is (21-10 = 11 msec). This optimization was done to avoid a split of up to a second between the times of the pictures taken by the cameras. The maximal split is possible, for instance, if one camera was started a second after the another camera. The first sessions we have made did not posses these characteristics thus a larger split (up to a second) may be present between pictures.

We were not able to share the same Wi-Fi network with the three cameras, or to make all the three cameras connect to the same network. Each of the cameras created a separate Wi-Fi network. There is a Wi-Fi Remote that can control up to 50 cameras. However, it was not possible to connect to the Wi-Fi remote by other devices. Thus, in order to download a set of pictures we should connect to each camera and download the last picture on the camera. This is not feasible as a picture is taken by each camera every two seconds, while connecting to a single camera takes between 380ms to 522ms and downloading a single image takes, between 4693ms to 5794ms. In addition, to avoid long transmission time, the resolution of the taken pictures is the lowest possible (5mp).

The previously noted time measurements were done when the phone was about 30cm away from the cameras and the Image analysis module was not running. Thus it is an optimal case. As this is a big overhead for our system that should handle a picture set in about 2 seconds we run our program with a mock that takes previously taken picture from a local folder and takes the phone sensors measurements from a log file recorded during the sailing session.

### 7.2 System Time and Power Consumption and Constraints

As previously noted, a picture should be taken by each camera once in two seconds. Thus, the computing time is very significant. The image analysis and heuristics components were tested on several systems, to check the time consumption of each component. The systems are described in Table 7.1.

During the testing the pictures were taken from a local folder, while phone sensors’ readings were taken from a log maintained during the relevant sailing session (or selected randomly). Table 7.2 summarizes the time each component detection takes. Each component is not run separately, but as part of the whole system, thus its time is dependant on the other components as well. Each component was measured on 3 different pictures sessions. On the server and on the laptop we did not exclude the times of reading the picture from the disk. However, on the smartphone reading from the SD Card takes significant amount of time (about 600msec). Therefore, we excluded this time when measuring. The total processing time was not measured on the smartphone as when the components run simultaneously, it is difficult to correctly exclude the reading time of each component. In addition, when performing the telltale detection on the phone, this processing takes tens and even hundreds of seconds. Therefore, this component was
<table>
<thead>
<tr>
<th>System</th>
<th>Type</th>
<th>Operating System</th>
<th>RAM</th>
<th>Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop</td>
<td>64-bit</td>
<td>Windows 7 Enterprise</td>
<td>4 GB (3.8 usable)</td>
<td>4 X Intel(R) Core(TM) i3 CPU M380 @ 2.53GHz</td>
</tr>
<tr>
<td>Smartphone</td>
<td></td>
<td>Android 4.0</td>
<td>1 GB</td>
<td>Quad-core 1.4 GHz Cortex-A9 CPU and a Mali-400MP4 GPU</td>
</tr>
<tr>
<td>Server</td>
<td>64-bit</td>
<td>Ubuntu 14.04 LTS</td>
<td>12 GB</td>
<td>4 X Intel(R) Xeon(R) CPU E5620 @ 2.40GHz</td>
</tr>
</tbody>
</table>

Table 7.1: Systems on which the image analysis and heuristics components were tested

not measured on the smartphone.

We also evaluated the communication time to a computer. The communication includes sending the telltale relevant data to a computer and getting an acknowledgment (HTTP 200 code) for it. We performed this measuring when the computer is on the same network and we sent only the amount of bytes for the relevant areas of the side pictures for processing. We actually sent the relevant subimage in JPG format to minimize communications costs. According to our measurements it takes on average: 0.21sec on average for a session of 1305 pictures. During the measurement the image analysis was on and so were the sensors. In addition, we should note that the reading time on the computer (the laptop described in table 7.1) took a significant part of this time (0.15sec on average on the same session). Therefore, if we would use a stronger computer this time might have been shortened. Nevertheless, in order to eliminate additional overheads, we deduce that at least the upper image analysis should be done on the smartphone to avoid sending the upper pictures, or parts of it to a remote computer.

The most costly part was the upper image lines detection, probably since we work with the largest area of the picture. We analyzed the different components and the time they require and saw that the most expensive parts in terms of processing time were the matrix operations over of the picture to detect the relevant area and the Hough Transform for lines OpenCV algorithm invocation. In order to minimize the time required by these parts, and as it is more complicated to optimize the OpenCV function invocation, we ported the matrix operations over of the image to RenderScript. RenderScript [Ren] allows to parallelize the computational tasks across the processors available on the device, including GPU.

The average time for processing the upper boat lines in the first session without RenderScript activated is 1620ms versus 1104ms when RenderScript kernel performs the transformation of the image (was tested with two consecutive runs of a single session). We wrote a RenderScript kernel for the sailor detection matrix operations as well as the relevant code in the boom detection. However, in these cases while running over 100 images we have not seen a noticeable difference in the detection time, while the tiller detection time became much worse. Thus, in the smartphone measurements presented in table 7.2 only the processing of the upper lines over the boat employ RenderScript.

In addition, we do not include in the measurement table the telltale detection time, as this component
takes about 100-200 seconds on the smartphone without porting any part of this code to RenderScript kernels. In addition, it slows down the other components. Therefore, we deduced that this component currently cannot be placed on the smartphone.

We conclude that the desired configuration of the program is that the upper pictures and the side elements except for the telltales are processed on the smartphone, while the telltale processing is done by a remote server which processes them within 545 msec on average. In addition, as noted in section 7.1, we could not test the system as a complete functioning system mostly due to the communication overhead of the communication model of the GoPro HERO3 cameras.

<table>
<thead>
<tr>
<th>Component</th>
<th>Laptop</th>
<th>Smartphone</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiller</td>
<td>49</td>
<td>410</td>
<td>59</td>
</tr>
<tr>
<td>Upper image lines</td>
<td>40</td>
<td>1057</td>
<td>37</td>
</tr>
<tr>
<td>Boom</td>
<td>205</td>
<td>746</td>
<td>98</td>
</tr>
<tr>
<td>Sailor</td>
<td>87</td>
<td>913</td>
<td>62</td>
</tr>
<tr>
<td>Centerboard</td>
<td>285</td>
<td>74</td>
<td>144</td>
</tr>
<tr>
<td>Wind indicator</td>
<td>242</td>
<td>241</td>
<td>103</td>
</tr>
<tr>
<td>Telltale</td>
<td>872</td>
<td></td>
<td>545</td>
</tr>
<tr>
<td>Total image analysis</td>
<td>1289</td>
<td></td>
<td>760</td>
</tr>
<tr>
<td>Heuristics</td>
<td>0</td>
<td>0.35</td>
<td>0</td>
</tr>
<tr>
<td>Total time</td>
<td>1293</td>
<td></td>
<td>766</td>
</tr>
<tr>
<td>Maximal time</td>
<td>8361</td>
<td></td>
<td>8251</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>966</td>
<td></td>
<td>951</td>
</tr>
</tbody>
</table>

Table 7.2: Time (msec) taken for the execution on each system in 3 different sessions

In terms of power consumption, we have tried to run a full session, in which the phone first activates the cameras, downloads a picture from each camera, processes 603 picture while sending the telltale relevant data to a remote server getting an HTTP OK response and while the other phone sensors are activated. The only factor we have omitted in this measurement is the GUI compass presenting, as we think that this factor can be removed in case it consumes much power, since the sailor will probably be busy with sailing and not looking in the application’s graphics. In addition, for technical reasons the speaker was mostly silent. The power consumption in this case was 19%. In addition, logging was activated during this session and logged for instance, the sensors reading. Therefore, we conclude that our application uses a reasonable amount of the phone battery and in terms of power consumption can be
used without special issues.

### 7.3 “Sailing Helper” Application Class Diagram

Figure 7.2 describes the main classes of the “Sailing Helper” application and the interaction between them. SailingHelperActivity is the main activity of the application. It holds a list of all the sensors available and starts them upon starting the application by the user. There are two types of sensors in the application, passive sensors which are represented by the ISensor interface and active sensors which are represented by IAlertingSensor interface which inherits ISensor. The passive sensors only collect data and provide this data when queried, while the active sensors send data upon receiving new data. ImageProcessingSensor interacts with the Camera via CameraCommunication. It also starts the cameras in the correct mode and stops them when needed. If we would have an option to obtain images from all the cameras, we would register as a listener to the CameraCommunication in order to get pictures from the camera, while the camera communication queries the cameras. Unfortunately, as we have already explained, in the time constraints we had and with the possible cameras network configuration, this was not possible. The ImageProcessingSensor must have also a class of ServerCommunication via which it can offload the “heavy” image analysis tasks to a server.

Each sensor accumulates data consisting of the data value and the time when it was acquired (represented in the class Measure). SensorData provides a HashMap of the Measures for the different parameters each sensor is measuring. An inner class of the SailingHelperActivity, ProcessMeasurements, which inherits from thread adds the data to an input class which accumulates all the data from the sensors and passed to a Heuristics class that decides what should be the output by the given input. The possible outputs are represented in an Output enum. The output, if exists, is then passed to a speaker that says the instructions.
Figure 7.1: Sequence Diagram of the Sailor Helper Application activation
Figure 7.2: Sailing Helper Class Diagram
Chapter 8

Related Work

Automatic trainers have been addressed in several works [EH98, LJJL05]. In [LJJL05], a trainer was created to the pool game while [EH98] presents a Fluency Pronunciation Trainer. The pool trainer in [LJJL05] also employs a camera and analyzes the pictures taken by it in order to assess the situation. For instance, to record the position of the balls and their movement. However, while the pool table is stationary and can be located in constant lightening, we use cameras and analyze the taken pictures in much harder conditions. In the pool trainer a reference image of the empty pool table is used. Unfortunately, in our system, as the boat constantly moves and a lot of elements such as the sail and the sailor move, we cannot rely on such a stationary reference picture. In our case, even a XOR between two adjacent pictures did not yield any valuable information we could use, due to multiple changes. Additional resemblance to this work is that both trainers provide spoken instructions.

In this work, we present an image analysis algorithm that was designed specifically for the described problem. Nevertheless, it uses techniques which were described in other ad-hoc solutions (for other problems). For instance, [DDD04, RNI12] propose image analysis solutions for detecting car license-plates and “reading” the content of the license-plates. These works utilize the Hough Line Transform algorithm, which was used in our work as well. In addition [RNI12] uses algorithms like Gaussian smoothing to eliminate noise and Canny edge detection and dilation in order to separate characters on the license plate. These techniques were used in our work as well. [KCK96] uses Hough Transform to find candidate lines of road lanes and thus uses a technique we have also used for solving another problem. [DDD04] uses contours detection algorithm which was widely used in our solutions. Both [DDD04, RNI12] use OCR techniques to identify the characters on the plate. We tried to use the OCR technique to identify letters on the sail, however, as have explained in this thesis, it proved to be unsuitable for our purposes. These known techniques are widely addressed in the literature, for instance in [Par10].

Autonomous sailing boats were presented in [oli, SRL+09]. These works employed various sensors like GPS and Anemometer. One of the solutions in [SRL+09] has also employed a mobile phone having a WI-FI, a Bluetooth, a GPS, and two Accelerometers. In [oli] the factors checked are very similar to what we check, i.e., they check the speed over water, if the sail is luffing, the boat orientation, etc. However, the sensors which are employed are different. For instance, piezoelectric deformation sensors are used for the sail and a pitot tube is used to measure the speed over water.

Sailing trainers are also addressed in several works. However, to the best of our knowledge, in all
these works the training is not done in real conditions at sea, but use virtual reality or simulations for the training session. Examples of such works are [Geo07, MSHB09]. There is a company “Virtual Sailing Pty Ltd” that has a Virtual Sailing VSail-Trainer [VSa]. A patent about sailing simulation was published in 1976 [Nis76]. Our solution is aimed to enable the sailor have a training session in real conditions in sea. To the best of our knowledge, no work has yet addressed a design and implementation of a sailing trainer.

Our trainer is developed for the Android operating system [But11], which is currently highly popular, with over a million applications available on Google Play and over 50 billion downloads until July 2013 according to [Goo]. A lot of innovative applications have been addressed in different works [SDT10, ST09]. Numerous fitness, brain and memory trainer Android applications are available and attempt to instruct the users [MCPM11]. The book [Sim] describes different Android applications, some of which are intended for sailors. However, none of the above mentioned applications attempts to instruct sailors in real life conditions.
Chapter 9

Conclusions

9.1 Discussion

There are several characteristics and aspects that are addressed in this work. First during the image analysis phase, we employed multiple ad-hoc optimizations, calibrations and heuristics in order to try to achieve better results. For instance, when searching for straight lines, what should be their length, to what extent can they be disrupted and once we found several candidates for some object how do we grade them to select the most probable. The optimizations and calibrations were discussed in Section 5.2.

Second, while performing the image analysis, we noticed that the lightening has a major effect on the algorithms and heuristics we employ and that the variation of the lightening is very high, as described in Section 5.2.9. Thus we employ 3 classes of lightening in the side pictures (side lightening) and 2 classes of upper lightening (which showed less variation). According to these classes the image analysis color related decisions are slightly changed, as the colors we expect are different.

An additional challenge is the versatile surrounding that contains a lot of shapes that look identical to what we are looking for. For instance, there are lines upon the sail that are easily confused with the telltales, given some light conditions. The surface of the boat might contain other straight lines, for example, when the sail’s shade is seen upon the boat. The surrounding we operate in is very challenging. We are not looking for objects on a constant blank white page, but in a surrounding that is constantly changing.

An additional optimization in which we put much effort during the image analysis is bounding the area that we address while detecting the different components. This allows us to speed up the processing while gaining accuracy. One of the factors on which we heavily rely is the location of the cameras, due to which we can approximately know where some objects of interest should be located. However, as the cameras were removed from the boat at the end of each sailing session and put back to the boat at the beginning of a new session, the zooming due to the cameras’ locations is changed from session to session. Therefore, this zooming is manually adjusted in each session.

An additional strategy that helped to eliminate errors both in the image analysis phase and in the heuristics is to check the previous frame. In the image analysis phase, in case of conflict we can take the result closer to the previous result, assuming that mostly the changes are not drastic. While in the heuristics we can give the sailor a chance to correct an error and only if the error remains in two adjacent
frames to warn the sailor. The above described strategy does not always show good results. For instance, we tried to employ it when looking for the boom, however, as described in section 5.2.2, this did not yield good results.

While planning the system we did not know what exactly will run on the smartphone and what will run on a remote server, thus the code of the image analysis and heuristics modules is portable and is ported as is from the smartphone to the server. The code is identical in all the devices it was run on. An additional aspect of the code is that it is modular, so that if we decide that several parts of the image analysis should be done on the smartphone while others are run on a remote server, the modules are easily detached.

Lastly, as we have several modules and the image analysis code is costly in terms of its running time, we try to make it as parallel as possible. We are utilizing the parallel capabilities of the devices the code is executed on.

9.2 Conclusions and Further Work

In this work, we have created a prototype of an autonomous sailing trainer. All components of the system were analyzed and their performance and limitations were studied. Yet, some technical limitations of the technologies selected and some performance limitations prevented us from presenting a complete live integrated system. These are mostly related to the current GoPro Hero wireless connectivity model and one aspect of the image analysis task. Hopefully, future generations of affordable wireless cameras would resolve the first issue while faster on-board GPUs on smartphones would help resolve the other. Also, smart cameras that can perform simple analysis and thereby only send small areas rather than entire images would also greatly help our task. We have employed several proven and known image analysis techniques in order to overcome the image analysis challenges we had, while tweaking them and their numerous parameters to fit our problems. We also tried to maximize parallelism and to order the processing models in an optimal way.

We deduce that a better image analysis solution is needed for the side frames which have much more lightening challenges. In particular, sometimes even the human eye cannot detect the object of interest. In addition, we had several sessions, in which due to weather conditions, such as strong winds the cameras moved, producing pictures without the object of interest. If keeping the approach of using image analysis only for detecting the objects of interest, some additional techniques might be considered in order to handle the varying lightening conditions. An idea that we have not eventually tried is to sample the color scheme of the sea or the sky and try to analyze the lightening condition. Moreover, one should keep in mind that the cameras should be attached strongly to the boat, so that they will not shift, even in harsh weather conditions.

One if the major image analysis bottlenecks we have encountered, both in terms of accuracy and runtime performance, is telltale detection. Another possible direction to explore here is the use of piezoelectric sensors placed on the sail. The tradeoff is that such sensors would complicate the deployment of the system, as they require some wiring. Yet, due to difficulty of the task using image analysis, it is a compromise worth considering. On the other hand, it is possible that improved camera technology that
can better handle changing lighting conditions might also help resolve this issue. Also, one could borrow from [SF11] their method for recovering the 3D shape of the sail by placing multiple distinct markers in a grid-like manner on the sail. That work, however, does not mention the run-time of the proposed technique. Also, it would complicate the deployment of the system considerably, as well as making the sail illegal to race with, i.e., such fitted sail can only be used for training.

Additional aspect that could be very useful for the trainer is to measure the wind speed, which we could not obtain in the desired accuracy by the audio. Moreover, the autonomous sailing trainer can be extended to support additional commands and additional input parameters can be considered.

We have also experimented with both a heuristics based expert system and a machine learning scheme for the recommendations part of our work. We have found that each was better in some aspects and worse in others, in particular since certain errors were not frequent enough for the learning mechanism to learn them. Consequently, our bundle can combine the outputs of both, each for the ones it excels in.

In an effort to boost performance on the smartphone, we have used the Renderscript kernel, which exploits the phone’s GPU and possibly other processing units. This has improved the performance of the upper image lines detection component, but resulted in worse performance when deployed in other components. Further investigation of how to boost performance by utilizing the GPU using other graphic kernels and explicit GPU optimizations is left for future research.

When examining trends in contemporary smartphones, the advancements in CPUs (more cores, each of which is faster), GPUs, and amount of RAM memory, we can observe that it is likely that in a few years the entire computational task is likely to be solvable on the phone, with no need to contact the server. However, at the moment, even the most advanced phone would still require some support from a remote server in order to complete the processing in time. Also, in order to improve accuracy, one might wish to employ better image resolution, which increases the image analysis time, so it is possible that offloading some parts to a remote server will remain a requirement even in the future.
Bibliography


[gop] GoPro Java API. http://goprojavaapi.blogspot.co.il/.


 컴퓨터ordan organising, we can insert various frameworks into the system to enable various components to be run on different hardware. This allows the system to be flexible and can run on different types of devices. In addition, the framework can use different components of the device, such as CPU and GPU, to optimize performance. However, the use of these components can lead to a decrease in performance if used improperly.

In summary, the conclusions of the work are that:

1. A variety of algorithms and parameters were used to analyze the images.
2. The most important aspect of image analysis is time consumption and accuracy of the results.
3. Additional sensors, such as piezoelectric sensors for measuring the state of the image, can be added. Of course, the addition of sensors will make the system more complex and may be delicate, but sometimes it is necessary to use external sensors.
4. The use of additional CPU units and different memory configurations can also improve the system's performance. It is possible to move parts of the system to different memory devices and check the effect on performance. Overall, the use of different frameworks and devices can improve the system's performance in various ways.
The algorithms developed are intended for objects that are very specific, with specific characteristics, and we used OpenCV and with various functions from this library, including a very specific algorithm which we fine-tuned. In addition, we developed libraries with various functions, where some parameters differ from those of the algorithm. The images were tested with a different set of 1,800 objects. In the analysis, the algorithm identifies objects in more pictures and analyzing images from the camera that goes to the decision-making stage.

The decision-making system, after it has identified the components of the input, sends the system to decide if it will or will not act. In order to make this decision, it uses a technique that was written in collaboration with the navigation personnel. Given the input, the system follows the rules of classification and determines the appropriate action. One of the options is the use of machine learning. We created a different system for each set of classified inputs and trained each system individually. Of course, if we do not have enough data for machine learning, we have to use another method.

The system works using a smartphone that is connected to the camera system. As the system runs, the system receives a still image from one of the cameras. In addition, the system separates the images from the camera and sends them to the system. After that, the system sends the picture to the smartphone. However, the cameras do not connect to a single Wi-Fi network. Therefore, the system needs to connect to each camera via Wi-Fi and send the picture to the smartphone. This process takes several seconds. Therefore, the main components of the system are tested separately. The system, in this case, is divided to the smartphone and the mobile device.

At a fixed moment on the ship's radar, without considering the position of the ship, or the ship's coordinates, or the direction of the wind, we can only estimate the distance between the ship and the object. But, while observing the ship's side view, we can see the ship's position relative to the radar. We can see the direction of the wind, which helps us estimate the wind's speed and direction. We can also use the ship's side view to estimate the wind's speed and direction. In addition, we can use a smartphone with a camera to estimate the speed and direction of the ship. We can also use GPS to estimate the speed and direction of the ship. In addition, we can use computer vision techniques to analyze images. First, because calculations take time, we can try to estimate the code as quickly as possible. In other words, we can use a combination of objects that are visible in a single image to solve the task. In addition, we can also use computer vision techniques to find the object. In this way, we can reduce the time taken to solve the task. In addition, we can use morphological and skeletonization techniques. In other words, we can use a fixed number of colors to search for objects. Therefore, we can perform operations on matrices to find the colors in the images. We then try to find straight lines that are parallel to the objects we are searching for.)

In conclusion, we can use computer vision techniques to estimate the speed and direction of the ship. We can also use a smartphone with a camera to estimate the speed and direction of the ship. In addition, we can use GPS to estimate the speed and direction of the ship. In addition, we can use computer vision techniques to analyze images.
הקדמה

העבודה Özcan שנוייב תואר ניסיון לעבר לפמッシュ מערכות משולבות ח@Inject של חזרה חונים ואיתות, גם מסתברות של מערכות היא לא פחוסי (ביהודי מבצעת חיבור) וтенוריים לדבש בפשיבת אמיתית, וסטצאי מחוון זה לא בטוח, ע {?י שמתנו עקר רישום ביחס בין מיקום לבין עליון. לא ניתן לשפה את השים דרך בפמッシュ בפמッシュ.

כן, אנו מתחמים בעבודה Özcan שנוייב, אך סירוב הממצאים סיכום בודד. על חיסון לגלם את חסידה לגלם

אתם פועלות בפמשות ממדוקדוק.LEC, על ומערכי להזיז קהל משמעון. בנסות, חי והמערכי מתארח ביער

לכבושות השינר, ממחית זו להזיז קבלת אוכלסיה י. ואכן, חי והמערכי להזיז פושוט כל

נתונים וחיסון, היא מעכרת مضמנים פעולות פgetRowות Wi-Fi כולל ידור בו פעולות GoPro HERO3 פעולות ידור

בשירות. חלחים מתארח בחר שידור קונ@GeneratedValue בנHôtel בשימוש בבר מרחוק.

המערכי מ단체 את מצבי החיסון והסירה בפשיבת הפעולה פקטור מלועה. עירוב חלומות מתבצע בכמויות קובליتعليم תdna התמחות

מתקפלות אלי התחפשות על התראה של מערכות עליה תחנתית, מחכים עלון מתכתלע עד ידי שילוב של

ויורסిטסוק ולארס חיותו.

ולא כי כל ימי פשיטים במעבז התחפשות ומקיונים, גם מﺿנים, מキッチン קילום במקומ ולעגי התנים, עירוב

רוחות חזרה עליון, מחכים לחרים ריבים שאז מצויצת את מיקומים במקומין יריבי בסאם את צורח וסים, עירוב

וירונבונד ואירוב ביצי מעברת. הערוכות את הביציועים ומעוגנות של הריבוכיס חווים. עי, את

בני משלוחים צמחיין אל הגלים לתוך מעברת שלמה שכתובות בברון מתן. א, עיינה להבנה יבשרא

לרבע מצויכות מייסי הזהות.

הקלט והפלט של מערכות

אומן, ישן 3 מצלמות המתק不开 על הסירה. מצלמה אחת ממקומ ברזים ציידים (מצלמות צייד). מצלמות ממקומ ברזים ציידים (מצלמות צייד).
המחקר בוצע בהנחייתו של פרופסור יהודה פרידמן, בפקולטת הנדסת מחשב.

הנודד

אני רצח להודות לענין זה, לישפוחתי ושלכל על שיעור או ייער לא יעי, לענין ענין זה. בפרט לדר' נעור אברוז, פרופ'닿ן יוגר פרידן, פרופ'-Identifier מיכאל לינדנבוים, פרופ'-Identifier שארל מרקוביץ. בנוסח ברצוני להודות לדר' נעור על העניין של ענין זה, לענין ענין זה.

אני מודע להכנת הטכניות שלענין זה, אוטונומיות ולענין זה, לענין ענין זה.
פיתוח מאמר שיניים אוטונומי

חיבור על מחקר

לשם מולי חלביק של הדרישות למעבר התואר
מניסים למדעי במדעי המחשב

ألנה בולשינסקי

הרמטכ"ל לפכנוניות - מרכז טכנולוגיות לישראל
נוכם התשע"ה חיפה למרץ 2015
פיתוח מאמץ שייט אוטונומי

אלה בולשנסקי