Adaptive Probabilistic Tack Manoeuvre Decision for Sailing Vessels

Sébastien Lemaire
sebastien.lemaire@soton.ac.uk

Yu Cao
Yu.Cao@soton.ac.uk

Thomas Kluyver
thomas@kluyver.me.uk

Daniel Hausner
dh4n16@soton.ac.uk

Camil Vasilovici
crv1g16@soton.ac.uk

Zhong-yuen Lee
leezhongyuen@gmail.com

Umberto José Varbaro
ujv1u16@soton.ac.uk

Sophia M. Schillai
sms4g13@soton.ac.uk

University of Southampton
Boldrewood Innovation Campus
Southampton, S016 7QF

Abstract

To move upwind, sailing vessels have to cross the wind by tacking. During this manoeuvre distance made good may be lost and especially smaller vessels may struggle to complete a tack in averse wind and wave conditions. A decision for the best tack manoeuvre needs to be made based on weather and available tack implementations.

This paper develops an adaptive probabilistic tack manoeuvre decision method. The order of attempting different tacking strategies is based on previous success within a timeout, combined with an exploration component. This method is successfully demonstrated on the 1m long sailing vessel Black Python. Four strategies for crossing the wind were evaluated through adaptive probabilistic choices, and the best was identified without detailed sensory knowledge of the actual weather conditions.

Based on the positive results, further improvements for a better selection process are suggested and the potential of using the collected data to recognise the impact of weather conditions on tacking efforts is recognised.

In: S. M. Schillai, N. Townsend (eds.): Proceedings of the International Robotic Sailing Conference 2018, Southampton, United Kingdom, 31-08-2018

1 Introduction

Current work on manoeuvre planning for sailing robots focuses on long term piloting from planning the actions until the next waypoint to routing a vessel over longer distances (Tynan, 2018; Langbein et al., 2011). When the routing towards a waypoint, all sailing vessels rely on manoeuvres to cross the wind: tacking, where the vessel turns with the bow facing towards the wind, or jibing, where the vessel turns with the bow facing away from the wind. Typically, sailing robots are controlled by using a rudder to control the heading of the vessel and then adjusting the sails based on the relative wind direction (Gomes et al., 2017) even if recent sail designs may include heading control in the sail (Augenstein et al., 2017). Vessel speed and boat drag are often recognised as a factor for successfully completing a tack (Jouffroy, 2009a; Jouffroy, 2009b). (Cruz and Alves, 2014) recognises that a sailing robot can get stuck facing into the wind, in sailing terms 'in irons', and suggests to recover from such a situation by increasing speed; thus gaining rudder control through letting the sails loose so the wind can push the boat backwards. (Tranzatto et al., 2015) studies how to perform fast and smooth tack manoeuvres using control system theory and compares 3 different rudder controls for tacking, however tack fails are not mentioned. Modeling a tacking manoeuvre could also be done to analyse the problem of tack failure. Several studies developed tacking simulators based partially on experimental measurements (Masuyama and Fukasawa, 2011; Roncin and Kobus, 2004; Spenkuch et al., 2010), however they are applied on large sailing boats where tack failure is not an issue, and is hence not discussed. When sailing the 1m long Southampton Sailing Robot, the Black Python, we found that the success and speed of a tack manoeuvre for such a small vessel not only depends on having sufficient speed and suitable rudder action to pass through the wind, but also on passing through the wave fronts pushed towards it by the wind. Where a human sailor would make choices about the sail and rudder settings based on speed and wave observations and experience, making small adjustment as the manoeuvre proceeds, implementing this process for a robotic sailor is challenging. Not only is it difficult to translate experience into software, but also the amount of sensor data that is required increases significantly compared to a dead-reckoning tack manoeuvre. As an alternative to fully measuring and considering all factors involved, the introduced system adapts to the wind and waves conditions by testing and evaluating several available tack methods.

This paper investigates a dynamic weighting approach to choose the best method to perform a tack in order to minimise the number of failed tacking attempts whilst having minimal sensor knowledge of weather and boat state. After introducing the Black Python vessel, its sensors and the software structure in the Systems section, we focus on the software components that control the tack manoeuvre, suggesting several tacking implementations and a tack weighting process based on previous successful and failed tack manoeuvres. The methods introduced are demonstrated on experiment results obtained in coastal waters near Southampton.

2 System presentation

2.1 The boat

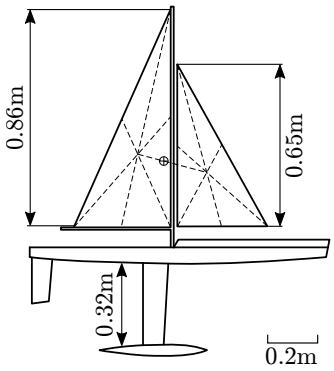
The Black Python, see Figure 1a, is a one-meter-long Lintel mono-hull sailing robot yacht of class IOM (International One Metre) designed by David Creed. This boat is designed for racing performances and is used in remote controlled regatta. Three different sets of sails with a sail area of 6000, 4100 and 2700 cm^2 can be used depending on the wind conditions. The hull's beam is 165 mm and the hull displacement is 4000 g. Minor changes have been made to fit wiring. The Black Python uses bulb keel that is 420 mm deep, and has a spade rudder for steering. Profile view of the Black Python is shown in Figure 1b.

2.2 Electronics

A Raspberry Pi 3 B (RPi) microcomputer is used as the main control board. It is powered by a 5V USB power bank. A foam stand and plastic box keep the RPi away from water that occasionally gets inside the boat.

A uBLOX MAX M8Q GPS unit gives the boat position and velocity, it communicates with the RPi via I^2C (Inter-Integrated Circuit). A conventional USB WiFi dongle is placed together with the GPS and a small IMU (Pololu AltIMU-10 v4) unit on the top of the mast. Inside the boat, an Xsens MTi 3 IMU (Inertial Measurement Unit) is used. It includes an accelerometer, gyroscope and magnetometer. The IMU is placed directly on top of the RPi. The yacht uses a custom made wind vane (Figure 1c). Two magnets are attached to the rotating part and a Pololu AltIMU-10 v4 is placed on the stator. The magnetometer on the IMU detects the change in the magnetic field as the wind vane is rotated by the wind, determining the wind direction relative to the boat.

(a) The Black Python in Southampton water



(b) Profile view showing the center of effort of the smallest set of sails (Papadopoulos, 2018)

(c) Custom made wind vane

Figure 1: The Black Python

A Futaba S3003 servomotor is used to drive the rudder and a HiTec 785 HB winch servomotor drives both mainsail and jib at once. A remote control receiver as well as a multiplexer are used to take control of the sailing robot in case of emergency or during the launch and recovery phases. The motors, multiplexer and RC receiver are powered by 4 AA batteries which are monitored with an Adafruit INA219 current sensor to ensure sufficient power for a remote controlled recovery.

2.3 Software

As mentioned in the previous section, the main computer of the Black Python is a Raspberry Pi. It runs the GNU/Linux distribution Ubuntu 16.04. The software is written in Python and utilises ROS (Robot Operating System[1], (Quigley et al., 2009)); a framework for writing robot software. ROS includes a collection of tools and libraries to simplify the task of developing complex behaviours. In the ROS ecosystem, the code is structured around scripts called nodes. Each node can send messages under a certain topic name: this is called *publishing*. Nodes can also listen for specific topics by *subscribing* to them. The entire software developed by the Southampton Sailing Robot Team is made available under the MIT free software licence[2].

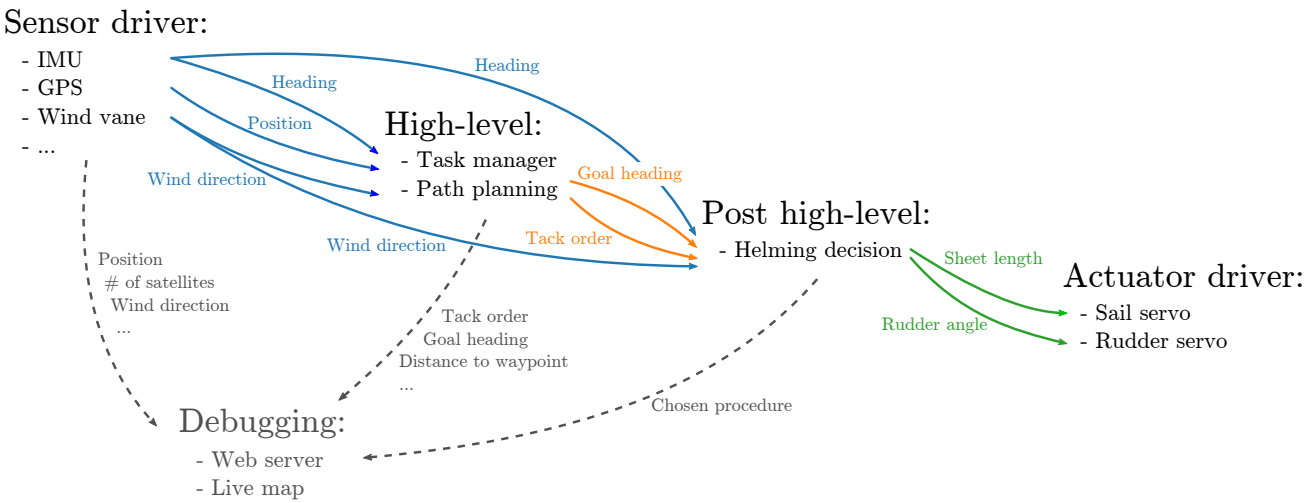


Figure 2: Structure of the code, arrows symbolise ROS messages passing

[1] `https://ros.org`
[2] `https://github.com/Maritime-Robotics-Student-Society/sailing-robot`

The software of the Black Python is structured as follows:

- *Drivers*: nodes talking directly to hardware components; this includes reading sensor data from the GPS, wind vane or obstacle avoidance camera as well as adjusting servo motor positions to set rudder angle and sail sheet.
- *High-level*: path planning nodes deciding when to switch tack and which heading to follow to complete a task. Available tasks are: reaching a waypoint, keeping a position, and avoiding an obstacle.
- *Post high-level*: the *helming node* converting the goal heading or tack order from high-level nodes into rudder angle and sail position. The *helming node* is described in depth in section 3.
- *Debugging*: nodes for visualisation of messages; from maps with waypoint and boat positions to graphical displays of angle information like heading, goal heading, and wind direction

Figure 2 illustrates the software structure, giving key nodes and the messages exchanging information between them. To achieve a particular assignment, the user configures the robot via parameter files. These include a list defining the tasks to run and task specific parameters like waypoint coordinates.

3 Post high-level: dynamic tack control

Weather conditions are sometimes not suitable for tacking on such small boats (dues to waves for example). In this situations a reliable approach is to jibe instead. The switch between tacking and jibing to cross the wind was implemented in the past on the Black Python using a user defined parameter set before each test. Whilst being a fail-safe method, we measured that jibing instead of tacking makes the vessel looses about 3m made good, when beating at 50 degree from the wind and with a speed of 0.75m/s. This leads to a loss of about 6 seconds per jibe, hence jibing should only happen when necessary. The post high-level was introduced to increase the choice of available manoeuvres alternatives to jibing and to automate the decision between all available tack and jibe manoeuvres.

The post high-level layer currently consists of the *helming node* alone, which operates between the low level drivers and the high level nodes. When the boat is not trying to switch tack, the rudder angle is set using a heading PID control. To set the sail sheet length, a predefined look up table containing the apparent wind direction versus the sail sheet length is used. When a tack order is given from the high-level nodes, the *helming node* is responsible for implementing a successful tack by changing the sheet length and rudder angle demands, choosing from a set of available procedures based on past events and a dynamic weight system with an exploration component.

In the context of the *helming node*, a procedure is the name given to a series of instructions that commands the sail and the rudder to perform a change of tack (jibe or tack). Several procedures are implemented:

- Basic tack (`BasicTack`): the rudder is set to its maximum position either on the port side if the boat was sailing on a port tack, or on the starboard side otherwise.
- Basic jibe (`BasicJibe`): the rudder is set to its maximum position in reversed compared to a tack. To help with bearing away the sails are sheeted out.
- Tack with sheet out (`TackSheetOut`): the rudder is set to its maximum position to perform a tack, and the sails are slightly sheeted out. On a conventional sailing boat the main sail tends to make the boat go more upwind, when the jib pushes the boat to go downwind. Sheeting out the jib can help with tacking, however on the Black Python both sails share the same control. This procedure hence tries to reduce the power in the jib by sheeting out a little both sails while conducting the tack.
- Tack with speed build up (`TackIncreaseAngleToWind`): to speed up the boat and gain momentum to aid passing the tipping point of the tack the boat will bear away for 5 seconds at 80 degrees from the wind. It will then perform an usual tack with setting the rudder to its maximum position.

The basic functioning of the *helming node* is as follows: Before each switch of tack, the procedure list is ordered by the time taken by each procedure in the past. The procedures in the list are tried in order until one succeeds to make the boat switch tack before a user defined timeout. After each procedure attempt, the time it took is recorded for future use to determine the order in the procedure list. A tack procedure is considered

a success if the time the procedure took in order to have the boat on the opposite tack (at an angle between 50 and 120 degrees relative to the wind) is bellow the user defined `timeout`. If a procedure fails it is placed further towards the end of the list by recording a value of 1.5 times the timeout. To ensure that all procedures are attempted, an exploration coefficient is considered as well.

Three user defined variables are used:

- `timeout`: timeout in seconds after which a procedure is considered as failed
- `ProcedureList`: initial order of the procedure list
- `Exploration coefficient`: probability (0 to 1) of picking an untried procedure instead of the top list entry

The `ProcedureList` is a python list of dictionaries, each element of the list has three dictionary keys: `Procedure` which is a pointer to the procedure class, `TimeList` a list of the time taken by the last 10 attempts of this procedure, and finally `InitPos` the initial position of the procedure in the `ProcedureList` as defined by the user.

Every time a tack is attempted, the `ProcedureList` is ordered based on weight. The procedure with the lowest weight will be tried first. The weights are computed as follows:

If the procedure have been tried in the past (ie. `TimeList` is not empty), its weight is the mean of the elements of the `TimeList`, in other words the average time the procedure took in the past. On the other hand, if the procedure has never been tried before, the `TimeList` is empty, then the `Exploration coefficient` is used. To know if the procedure will be picked by the exploration, a random number between 0 and 1 is generated. If it is above the `Exploration coefficient` no exploration is done. A combination of the `timeout` and the `InitPos` is given as the weight. This places the procedure between already attempted procedures, before the failed ones but after the succeeded ones whilst keeping all unused procedures in order of the initial list. Otherwise, if the randomly picked value is bellow the `Exploration coefficient` the procedure is placed at the top of the list by giving it a random weight between 0s and 0.1s (the random value ensures an arbitrary selection between methods picked by the exploration coefficient). The computation of the weight is summarised in Algorithm 1.

Algorithm 1 Function to get the weight of each procedure

```
1: function getWeight(procedure)
2:     if procedure.TimeList not empty then
3:         return mean(procedure.TimeList)
4:     else
5:         if random(0,1) < explore_coef/number_of_untested_procedures then
6:             return random(0, 0.1)
7:         else
8:             return timeout + 0.01*procedure.InitPos
```

Once the `ProcedureList` is ordered, it will not be reordered until the next high-level command to change tack. The list entries are attempted in order until a procedure successfully completes before the timeout. If all elements of the `ProcedureList` are tried without a success, the procedure selection will continue with the same list, beginning at the top entry.

3.1 Example of run

In this section a step by step example of a fictional run is described and illustrated in figure 3. The wind is coming from the North, and the boat is beating upwind. The user defined parameters are as follows:

- `Timeout`: 15s
- `ProcedureList`: [`BasicTack`, `TackSheetOut`, `BasicJibe`]
- `Exploration coefficient`: 0.3

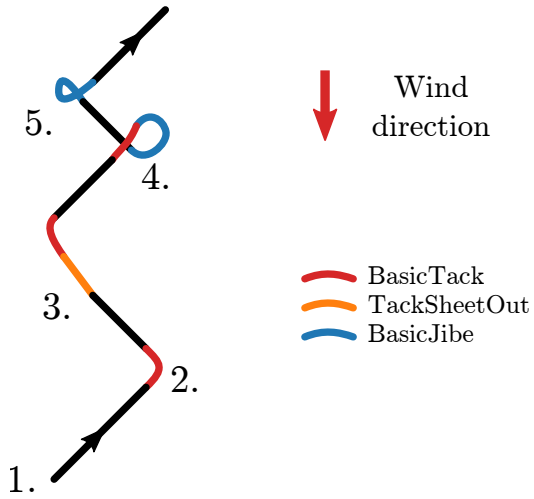


Figure 3: Fictional example of run with the *helming node* procedure picking

1. The boat starts sailing on a port tack, close hauled.

2. The *helming node* receives a tack change command from the high-level nodes. The `ProcedureList` is sorted. None of the unused (all) procedures is moved to the top of the list through the `Exploration coefficient`, hence the `ProcedureList` is as defined by the user: [`BasicTack`, `TackSheetOut`, `BasicJibe`]. The `BasicTack` is tried, it succeeds in 7 seconds. The boat continues on a starboard tack.

3. The high-level commands the *helming node* to change tack. The `ProcedureList` is ordered. This time the weight of the `BasicTack` is 7, making it the node with the fastest average time. The `Exploration coefficient` places the `TackSheetOut` procedure first in the `ProcedureList`: [`TackSheetOut`, `BasicTack`, `BasicJibe`]. The `TackSheetOut` is tried, but does not succeed before the timeout of 15 s. A value of 22.5 (1.5 times the timeout) is stored in the `TimeList` for the `TackSheetOut`. The next procedure on the list, the `BasicTack`, is tried and succeeds in 8 seconds. The boat continues on a port tack.

4. The high-level commands the *helming node* to change tack. The `ProcedureList` is ordered, the `Exploration coefficient` causes no re-ordering. The `ProcedureList` is: [`BasicTack`, `BasicJibe`, `TackSheetOut`]. The `BasicTack` is tried and fails, the next procedure (`BasicJibe`) is tried and succeed in 9 seconds. The boat continues its course.

5. The high-level commands the *helming node* to change tack. The `ProcedureList` is ordered, no re-ordering is caused by the `Exploration coefficient`. The `BasicTack` has a weight of 12.5 (mean of 7, 8 and 22.5), `TackSheetOut` has a weight of 22.5 (failed once) and `BasicJibe` has a weight of 9. The order is hence [`BasicJibe`, `BasicTack`, `TackSheetOut`]. The `BasicJibe` is tried and it is a success.

3.2 Discussion on parameter selection

The user defined parameters were purposefully kept at a minimum and chosen to have an easily understandable meaning. The exploration coefficient may however demand some experience of the boat behaviour to be set properly. A low exploration coefficient should be used when the user is confident with his ordering of the procedure list or when he knows that very few tacks will be performed during the test. Hence finding the best possible manoeuvre is not as rewarding as finding a working manoeuvre. On the other hand when a larger number of tacks are to be expected, setting the exploration coefficient higher will help finding the most performant method. The timeout should be set at the minimum value that allows ones boat to perform a tack manoeuvre in the expected (or all) weather conditions. A too low timeout will lead to the helming node cycling through the procedures and keeping failing when a too high value will make the boat loose time when trying a procedure for the first time.

4 Experiments

The experiment was conducted on the sea near Southampton Sailing Club shown in Figure 4. Wind direction gradually shifted from south-east to south. The averaged wind direction according to the wind direction data collected from the wind vane on the boat is shown in Figure 4a. Averaged wind speed was 4 knots with gusts at 5 knots. Small waves with height of 15 - 20 cm have been observed during the test.

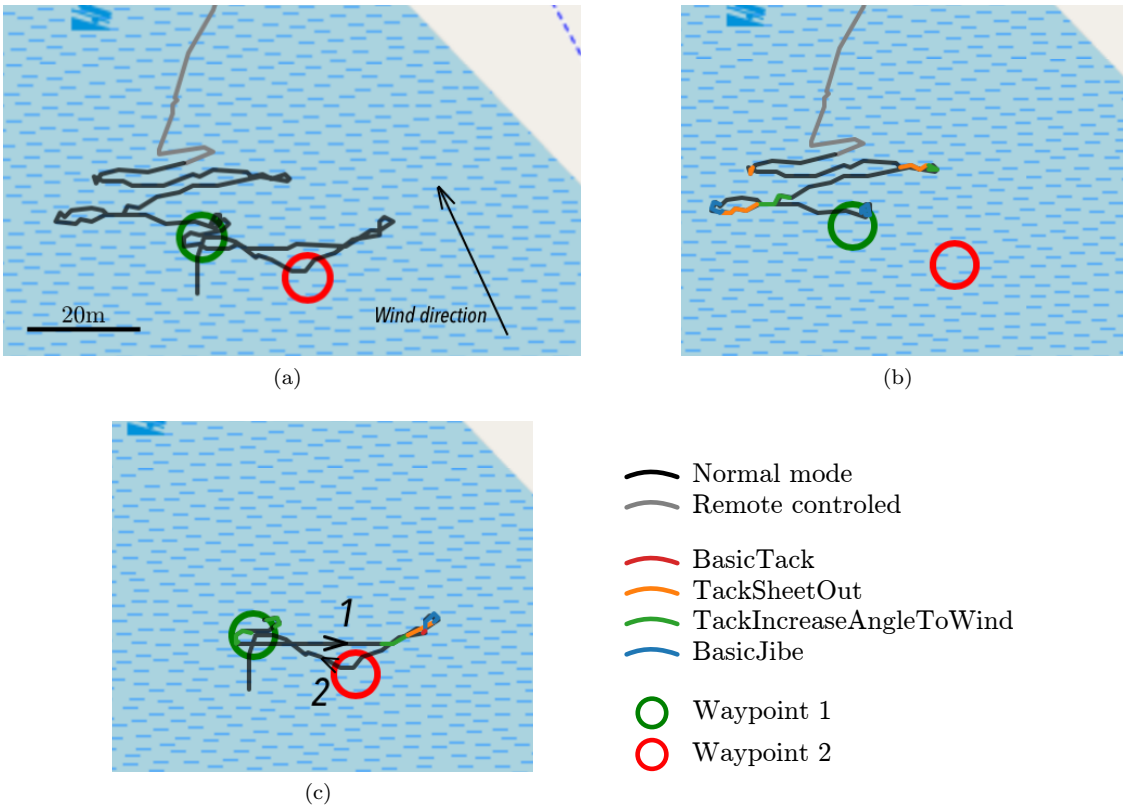


Figure 4: Experiment results in Southampton water

During the experiment, the boat was programmed to sail between two waypoints separated by 20m shown as a green and red circle on Figure 4. The Black Python was released from a runway on north of the visible map. To avoid being washed back by the waves the boat was first navigated into the ocean by a remote control. Once the boat was further away from the shore, the autonomous mode was activated. It tried to reach the first waypoint using the *helming node* described in previous section. An acceptance radius of 1.5m was set for all waypoints, parameters used in this experiment were:

- `timeout`: 30s
- `ProcedureList`: [`BasicTack`, `TackSheetOut`, `TackIncreaseAngleToWind`, `BasicJibe`]
- `Exploration coefficient`: 0.3

Experiment results are shown in Figure 4. In the first leg, three tack manoeuvres were made to reach the waypoint. As shown in Figure 4b, the first tack was done with a `TackSheetOut` procedure. Whilst being the second element in the initial `ProcedureList`, it was tried first because during the manual controlled phase the *helming node* was still running and a `BasicTack` failed, moving this procedure at the end of the list. This behaviour is not intended and will be fixed. The `TackSheetOut` succeeded in 9s. For the second manoeuvre, `TackSheetOut` was tried first and failed, `TackIncreaseAngleToWind` was then conducted and succeeded in 19s. For the next tack, `TackIncreaseAngleToWind` was tried first. It however failed, the next element in the sorted list now being `TackSheetOut`. This procedure was tried and also failed, finally a `BasicJibe` was conducted with success in 19s.

After getting to the first waypoint, the boat sailed towards the second waypoint as shown in Figure 4c. Two `TackIncreaseAngleToWind` were performed and succeeded. For the next tack manoeuvre, all four procedures were tried: first `TackIncreaseAngleToWind` did not manage to perform the tack leading to the boat bearing away, then `TackSheetOut` was tried without success. Later `BasicJibe` was tried, the boat managed to switch tack, however, the jibe did not finish on time and switching to `BasicTack` was needed to finalise the manoeuvre. The last leg from the second waypoint to the first one is downwind, the *helming node* did not start any procedure and the boat sailed in a straight line. After the boat reached the first waypoint again, wind condition stopped us from doing any further repeating tests.

On this day, the low wind speed made it particularly difficult leading to a lot of failed manoeuvres. Here the exploration never picked a random untried procedure because all procedures were tried in the first three tacks. With such low wind conditions, increasing the timeout might help reducing the number of failed manoeuvres because regardless of the method the boat is very slow to switch heading. This test demonstrated good functioning of the *helming node* and results show that in such weather jibing or taking up speed by bearing away first helps switching tack.

5 Concluding remarks and future work

Tacking on a small sailing boat is a delicate manoeuvre, the method presented here efficiently identifies a good strategy to perform a tack. The key aspect of the *helming node* framework and decision system is its simplicity; it does not rely on any additional hardware or even complex data processing but only on sensors that are already widely used on robotic sailing boats (heading and wind direction). It makes it easy to implement and to debug. From an user point of view, the simplicity of the implementation is also visible by the limited number of parameters and their physical meaning. Only 3 user defined parameters are needed (a timeout, a sorted list of procedure and an exploration coefficient) and all of them have an easy to understand meaning, no extensive knowledge of the boat behaviour or the weather conditions is needed.

The *helming node* system was successfully demonstrated, but it can still be improved on several aspects. If all the manoeuvres of the `ProcedureList` fail, this can mean that the selected timeout is too short. Instead of rerunning the list with the same parameter, increasing the timeout automatically would be judicious. Additional tack procedures could further consider the sea state, for example by timing the tacks based on the position of the boat on the waves. Although some work to detect wave period has been done by the Southampton Sailing Robot Team, more tests are still needed to refine the method and integrate it as a procedure. Lastly, for now only the time taken by a procedure is measured to assess its performances. To more accurately consider the distance lost during the jibe manoeuvre combining the time with the distance gained towards the next waypoint, for example, could be an improvement of the *helming node* weighting process. Also the method is currently not suitable for long term tests where weather conditions might change over time. An improvement of the weighting that includes an aging parameter would be preferable in this case.

After performing more tests with this new system, a better understanding of each procedure will be gained and more precise and general conclusions concerning the best way to perform a tack in a specific weather condition can be drawn.

References

Augenstein, T., Singh, A., Miller, J., Pomerenk, A., Dean, A., and Ruina, A. (2017). Using a controlled sail and tail to steer an autonomous sailboat. In *Robotic Sailing 2016*, pages 91–103. Springer.

Cruz, N. A. and Alves, J. C. (2014). Navigation performance of an autonomous sailing robot. In *Oceans-St. John's, 2014*, pages 1–7. IEEE.

Gomes, L., Costa, A., Fernandes, D., Marques, H., and Anjos, F. (2017). Improving instrumentation support and control strategies for autonomous sailboats in a regatta contest. In *Robotic Sailing 2016*, pages 45–56. Springer.

Jouffroy, J. (2009a). A control strategy for steering an autonomous surface sailing vehicle in a tacking maneuver. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 2391–2396. IEEE.

Jouffroy, J. (2009b). On steering a sailing ship in a wearing maneuver. *IFAC Proceedings Volumes*, 42(18):26–31.

Langbein, J., Stelzer, R., and Frühwirth, T. (2011). A rule-based approach to long-term routing for autonomous sailboats. In *Robotic Sailing*, pages 195–204. Springer.

Masuyama, Y. and Fukasawa, T. (2011). Tacking simulation of sailing yachts with new model of aerodynamic force variation during tacking maneuver. *SNAME Journal of Sailboat Technology*, 1.

Papadopoulos, I. (2018). Techniques to improve the tacking performance of model scale robot yachts. Technical report, University of Southampton.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source robot operating system. *ICRA workshop on open source software*, 3(3.2):5.

Roncin, K. and Kobus, J.-M. (2004). Dynamic simulation of two sailing boats in match racing. *Sports Engineering*, 7(3):139–152.

Spenkuch, T., Turnock, S., Scarponi, M., and Shenoi, A. (2010). Real time simulation of tacking yachts: how best to counter the advantage of an upwind yacht. *Procedia Engineering*, 2(2):3305–3310.

Tranzatto, M., Liniger, A., Grammatico, S., and Landi, A. (2015). The debut of aeolus, the autonomous model sailboat of ETH zurich. In *OCEANS 2015-Genova*, pages 1–6. IEEE.

Tynan, D. (2018). An attractor/repellor approach to autonomous sailboat navigation. In *Robotic Sailing 2017*, pages 69–79. Springer.