Kinematic Constraints and ns-3 Mobility Models: the AUV Issue

Matteo Franchi
Dpt. Industrial Engineering
Università di Firenze
Firenze, Italy
matteo.franchi@unifi.it

Tommaso Pecorella
Dpt. Information Engineering
Università di Firenze
Firenze, Italy
tommaso.pecorella@unifi.it

Alessandro Ridolfi
Dpt. Industrial Engineering
Università di Firenze
Firenze, Italy
alessandro.ridolfi@unifi.it

Romano Fantacci
Dpt. Information Engineering
Università di Firenze
Firenze, Italy
romano.fantacci@unifi.it

Benedetto Allotta
Dpt. Industrial Engineering
Università di Firenze
Firenze, Italy
benedetto.allotta@unifi.it

ABSTRACT

Recently there has been a renewed interest in ns-3 as a tool for Underwater Acoustic communications, with the integration of World Ocean Simulation System (WOSS, [4, 6]) into ns-3. However, the current implementation of ns-3 does not provide specific models suitable for AUVs (Autonomous Underwater Vehicles) mobility. An old proposal is available, made by Andrea Sacco during his Google Summer of Code (GSoC) 2010 project. However, the code has never been integrated into ns-3. In order to simulate the communications of AUVs, it is mandatory to rely also on simple and effective mobility systems, where the kinematic constraints of the node are taken into account. The requirements of a mobility model for AUVs is to be able to take into account the kinematic model of the real device and to set up a feasible path between two (or more) points. This paper presents a new extensible architecture based on kinematic models, which greatly simplifies the simulation complexity.

CCS CONCEPTS

• **Networks** → **Network simulations**; **Network mobility**;

KEYWORDS

ns-3, Mobility Models, AUV

ACM Reference format:
Matteo Franchi, Tommaso Pecorella, Alessandro Ridolfi, Romano Fantacci, and Benedetto Allotta. 2017. Kinematic Constraints and ns-3 Mobility Models: the AUV Issue. In *Proceedings of the 2017 Workshop on ns-3, Porto, Portugal, June 2017 (WNS3 2017),* 7 pages.
DOI: http://dx.doi.org/10.1145/3067665.3067673

WNS3 2017, June 2017, Porto, Portugal

978-1-4503-5219-2/17/06…$15.00
DOI: http://dx.doi.org/10.1145/3067665.3067673

1 INTRODUCTION

AUVs are increasingly used for data collecting (e.g. optical images, acoustic images, measurements about water properties and quality, etc.) in industrial, military, environment protection, geology, biology, and, recently, submarine archaeology.

An AUV can be used as a single unit, performing most of the task in a given operation. However, teams of AUVs, each of them designed to accomplish a particular task, can be used to optimize the investment and operation costs and to quickly execute a complex mission [2].

A single AUV needs a communication channel to be monitored by the command and control system, and to offload the gathered data once the mission is over. On the opposite, communications play an important role in AUVs teams, because there is the need to coordinate the single units sub-missions in order to perform a larger scope goal, and the instantaneous data gathered by a node can require changes to the overall mission plan. As an example, a swarm of AUVs could be used to track a pollutant or to inspect a submarine archeological site. Upon finding an 'interesting' point, an AUV can request other, more specialized, AUVs to assist.

Unlike terrestrial networking technologies, underwater communication systems are in early development stages. As a matter of fact, acoustic channel models are more complex and acoustic modem devices are often using proprietary protocols, making it difficult to perform simulations. Recently a new standard named JANUS [7] has been proposed for underwater communications, but it is relatively slower than the proprietary protocols used by many acoustic modems.

Beside the actual communication standard used by the devices, it is important to create an accurate mobility model suitable for AUVs in order to have simulation scenarios that are close to reality. The current implementation of ns-3 does not provide a model, or a support altogether, for mobility-constrained devices.

This paper presents a new kinematic model for AUV. It takes into account the capabilities of a mobile node to perform straight-line movements and rotations, along with assert conditions on meaningful physical quantities. The proposed solution performs a good trade-off between computational complexity and accurate modeling and, it can be extended to other mobile node types, e.g., other AUVs, etc..

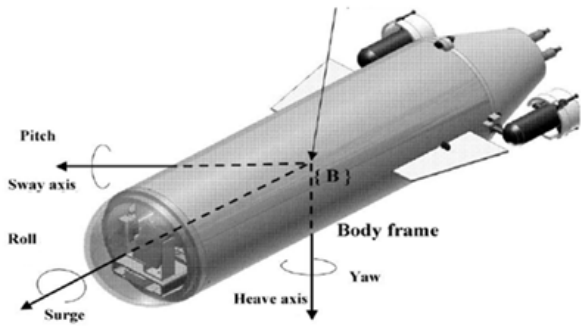

Figure 1: Generic AUV.

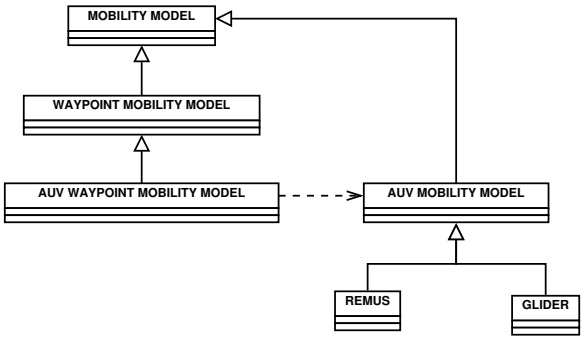

Figure 2: A. Sacco's AUV mobility models (simplified UML).

The rest of the paper is structured as follow. Section 2 describes the existing AUVs mobility models, along with their limitations. Our proposal is presented in Section 3. Section 4 shows some simulation examples their results. Lastly, Section 5 draws the conclusions and discusses limitations and future developments.

2 MOBILITY ISSUES AND STATE OF THE ART

This section briefly describes the AUV mobility model requirements and the ns-3 mobility limitations, both the ones already included in ns-3 and the ones proposed by Andrea Sacco in 2010.

2.1 AUV Kinematics

An AUV can be represented by a solid object moving in a 3D space (see Figure 1). It is important to notice that, depending on the vehicle, a given movement could be impossible to perform directly. As an example, it is safe to assume that a movement along the *Surge* axis is always possible, but a rigid translation along the *Heave* and *Sway* axes requires additional thrusters. Similarly, a rotation along the *yaw* axis requires two thrusters (one on the front and one on the back of the AUV), while a change in the Pitch angle can be performed with a buoyancy control or through diving planes.

The propelling type, the allowed movements along various axes, and the limitations on some angular quantities are the most important elements differentiating the various AUV types.

It must be noted that the differences do not impact (usually) if an AUV can move from point A to point B, but it changes the intermediate points used to reach the final destination. In other terms, two different AUVs will use different paths to reach the same destination.

2.2 Mobility in ns-3

In the current ns-3 release, the *MobilityModel* tracks the node position and its time evolution with respect to a Cartesian coordinate system with only three coordinates: (x, y, z). I.e., the mobile object is represented as a body-less and dimensionless point in space.

Unfortunately, this kind of representation is not sufficient when the object *orientation* (i.e., its *attitude*) is important as well. An AUV, for example, may need to rotate around itself or modify the pitch angle (see Figure 1) when changing direction. Summarizing, the three coordinates system is not sufficient to model an AUV. As a result, AUVs will need an *extended* coordinate system, considering not only the actual node position but also the head direction (at least).

It must be pointed out that also the *Antenna* module has a similar issue. The antenna "orientation" (for non-isotropic antennas) is not automatically updated when the node changes its course. This could be beneficial (for gyroscopically stabilized antennas) or not (for fixed mounted antennas).

2.3 Current AUV Mobility Models

The aim of this work is to enhance the current AUV mobility models developed by Andrea Sacco and not yet included in ns-3. The class diagram of the models is shown in Figure 2. Two types of AUVs are considered: *Remus* [1] and *SeaGlider* [5].

It is evident how the design in Figure 2 has some drawbacks. First and foremost, the class *AuvWaypointMobilityModel* has a member variable of type *AuvMobilityModel* (for real one of its sub-classes). Both *AuvWaypointMobilityModel* and *AuvMobilityModel*l are derived from the *MobilityModel* class, which is the one used by a node to know its kinematic state.

This structure is not optimal because both *AuvWaypointMobilityModel* and *AuvMobilityModel* derived classes have their own node status (i.e., its extended coordinates), and they will need to have them both updated and in sync.

For real, the instance used by the node is the *AuvWaypointMobilityModel*, while the *AuvMobilityModel* derived class is used by the owning instance to verify the movement feasibility (i.e., if the movement is allowed). It is worth noticing that, theoretically, the classes derived from *AuvMobilityModel* could be used directly by a node.

The core of the system is based on the class *AuvWaypointMobilityModel* which permits to go from point A to point B. The class allows defining a list of intermediate points to be used, along with the time they need to be reached. However, all the points pairs have to satisfy the kinematic constraints defined by the *RemusMobilityModel* or *GliderMobilityModel*. Moreover, if a constraint is not met, an assert is raised, and the simulation is blocked. As a consequence, an application trying to control the node movements should know in advance the node kinematic constraints, making it redundant to check them later.

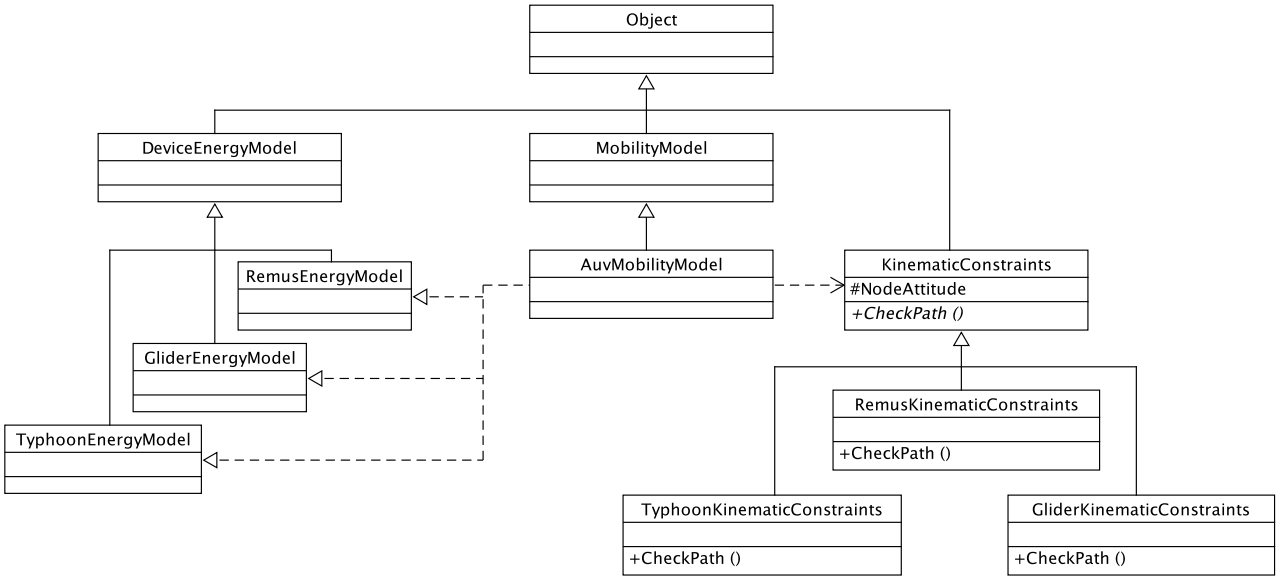

Figure 3: The proposed mobility model.

Lastly, *MobilityModel* class presents the defects outlined in Section 2.2: only the three spatial coordinates (x, y, z) are considered, i.e., the robot is seen as a material point: the node attitude (pitch, yaw and roll angles) is not taken into account.

3 NEW AUV MOBILITY MODELS

This section describes the new features introduced by our proposal and it illustrates in detail the system for two particular types of AUVs called *Typhoon* and *MARTA* which have been developed at the University of Firenze, Italy (see [2, 3]). In Section 3.3, a brief sketch about other classes of AUVs is given.

3.1 New Features

The structure of the system in Figure 2 has been revised and the new class diagram is shown in Figure 3.

The main idea is that there should be an AUV-specific class defining the movement capabilities and able to calculate a set of *intermediate points* between two given Start/Stop points.

The *KinematicConstraints* class derives directly from *Object*. It is an abstract class and it defines the interface to query a concrete class the intermediate path. There are three different AUVs concrete constraints classes (at the moment):

- *TyphoonKinematicConstraint*: for Typhoon and MARTA class AUVs.
- *RemusKinematicConstraint*: for Remus class AUVs.
- *GliderKinematicConstraint*: for SeaGlider class AUVs.

Each class implements the *KinematicContraints::CheckPath* function (an abstract, virtual function). The input parameters are two points and the time the node should be in each one. I.e., defined a point in space-time P as (x, y, z, T), the function accepts two points: $P_{\text{Start}}, P_{\text{Stop}}$. The function output is either a list of intermediate points $P_i, i \in 0..n$, or an assert when no feasible path can be found. The returned points are then added by the caller (typically *AuvWaypointMobilityModel::AddWaypoint*). In this way, the responsibility to choose the AUV path is logically split into two components: the first is an high-level decision (where the AUV should pass by and when), and one low-level one (build a feasible path between each pair of points).

With the new design, it is possible to consider both the 3D kinematic limitations (e.g., maximum pitch) and the motor capabilities (i.e., maximum speed along any axis). Practically, the asserts are hit only if the AUV reaches its limit operational depth or if there is no time to reach the end point, i.e., the required speed to too high.

Moreover, the KinematicContraints derived classes can memorize the previous node movement *direction*, in order to expand the Mobility model representation and apply the needed rotations to the AUV before an actual movement is performed.

It is possible to calculate the node attitude from the movement direction, provided that we neglect external forces (e.g., currents). In the present work, this assumption is met. As a consequence, the KinematicContraints class tracks the node attitude. A better solution is outlined in the conclusions. In order to avoid ambiguities at the simulation start, we decided that any AUV will begin the simulation being pointed toward the intended destination with a pitch and roll set to zero. This is, of course, arbitrary.

The actual mobility model used by the *AuvWaypointMobilityModel*, i.e., the one that links the intermediate points, is a constant velocity mobility model. It is possible to overcome this limitation by adding more intermediate points (i.e., splitting the acceleration and deceleration phases is constant-speed segments) or by extending the model to take into account also this parameter.

As a final remark, the new model allows considering the time required to reach a correct attitude between two movement segments

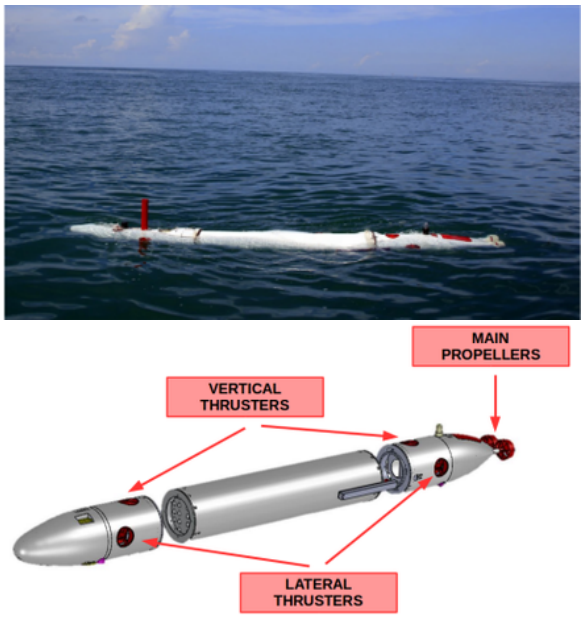

Figure 4: The Typhoon AUV.

(i.e., to rotate the AUV along its axis). Summarizing, we think that the split between the movement mode and the constraint model is a flexible solution to the AUV mobility problem and that it could be also extended to other mobile nodes.

3.2 Implementation Details

As explained in the previous section, the functions overriding *KinematicContraints::CheckPath* must take into account the constraints of the AUV. Therefore it differentiates, at the kinematic level, the behavior of the vehicle.

In this section, we will explain in detail the actual movement pattern of a particular type of AUVs (*Typhoon*, see Figure 4), which has been built at the MDM Lab, Dpt. of Industrial Engineering of University of Florence. This class of AUVs has the following properties:

- The robot can move along all the axes (surge, heave, and sway) and can rotate around the heave axis (yaw angle).
- The pitch angle is usually set to zero, whereas the roll angle is neglected.
- The robot has the following technical limitations:
  - Max speed along the surge axis (horizontal) = 2.5 m/s;
  - Max speed along the heave axis (vertical) = 0.26 m/s;
  - Max angular speed around the heave axis = 0.3 rad/s;
  - Max operating depth = 300 m

The *MARTA* class is almost identical, with a limitation of 1.5 m/s on the surge axis speed and 100 m maximum depth. It is worth to point out that these are not limitations of the model, but they represents the actual behavior of the vehicle. The steps below are also shown in Algorithm 1.

In the following, unless otherwise stated, we will indicate with the word *'point'* a 3D point plus a time. Given two points called as *'start point'* (the first one) and *'stop point'* (the last one) the *TyphoonConstraints::CheckPath* function works as follow:

Algorithm 1: TyphoonConstraints::CheckPath

```
Data: START(x, y, z, t), STOP(x, y, z, t)
Result: Intermediate waypoints
Initialization: AUV aligned toward the next waypoint
// Check the requirements
if stop.position.z < 0 || stop.position.z > maxDepth then
    abort;
// Reach the right depth
if stop.position.z != start.position.z then
    dive / rise;
    if actual.position == stop.position then
        if required speed > max vertical speed then
            abort;
        else
            return
Set dive / rise speed to max;
Calculate intermediate point P1;
// Move to the end point
if Required yaw is different from actual yaw then
    Rotate;
    if actual.time > stop.time then
        abort;
Calculate intermediate point P2;
Move to the end point;
if required speed > max horizontal speed then
    abort;
return P1, P2
```

As a preliminary step, a check about the general feasibility of the *stop point* is performed, i.e., if the required depth is larger than the maximum operating depth or it is negative. In both cases, the run is stopped with an error.

The other steps represent the core of the function.

The first operation is to reach the desired depth, i.e., go from *start point* depth to *stop point* depth. This is performed at the maximum allowed vertical speed. Note that the choice to use the maximum vertical speed is arbitrary, but it has been confirmed by the AUV designers that this is the "standard" procedure.

Afterward, unless the *stop point* was immediately below the *start point*, it is necessary to perform a horizontal plane movement. In this case, the movement is split between two sub-steps:

(1) The AUV rotates along the yaw angle in order to head toward the destination, and
(2) The AUV moves linearly until it reaches the *stop point*.

The rotation is performed at the maximum rotation speed (also in this case it is an arbitrary choice). On the opposite, the last movement is proportional to the time left to reach the *stop point*.

Each movement is associated with an execution time (including the rotation), and if the time required to perform an operation at the maximum speed exceeds the available time, an assert is raised.

Figure 5: Remus class AUVs.

Figure 6: SeaGlider class AUVs.

In other words, the *TyphoonConstraints::CheckPath* function makes the AUV rise/dive if necessary, then it rotates (if necessary) and it moves on a plane towards the *'stop point'*.

The *TyphoonConstraints::CheckPath* returns the intermediate points, i.e., the one when the AUV did reach the right depth (if any) and the point when the AUV did reach the right orientation (if any).

3.3 Remus and SeaGlider Mobility Models

The *Remus* [1] AUV (Figure 5) is quite different from *Typhoon*. The first, very noticeable, thing is that the AUV lacks vertical (and side) thrusters. As a consequence, it is unable to perform a direct descend or ascend (motion along the heave axis, see Figure 1). On the contrary, it is able to modify its pitch angle and proceed in a diagonal direction.

The intermediate points returned by *RemusConstraints::CheckPath* are similar to the ones produced by *TyphoonConstraints::CheckPath*, with one big difference: the descent/ascent is performed diagonally.

The pitch adjustment speed has been guessed because *Remus* changes its pitch by changing its buoyancy and by moving its direction fins and activating the thrusters.

The yaw rotation, on the contrary, is a misbehavior of the model. A real *Remus* AUV can not rotate along the yaw angle: the AUV would perform this movement by doing a small circle because *Remus* AUVs does not have side thrusters.

The *SeaGlider* [5] (Figure 6) design is even more different: the AUV moves taking advantage of its buoyancy, therefore the path towards the subsequent points is characterized by an up/down behavior. Like for the *Remus* case, the yaw rotation is not realistically modeled.

3.4 Energy Models

Energy models were already present in Andrea Sacco's early models. We added the relevant class for *Typhoon*, and we slightly adapted all the classes to the new design.

All the energy consumption is calculated according to the node movements. As a consequence, the energy update is triggered by the *CourseChange* Trace, or by an explicit request to the energy model. The main energy parameters for the three AUV types are in Table 1.

The *SeaGlider* energy consumption parameters are not shown because they are dependent on the buoyancy and required vertical speed. The used parameters and formulas can be found in [5].

4 SIMULATION RESULTS

We used a simple 2-points scenario to evaluate the different AUVs behavior and the correctness of the simulations. The AUVs are required to reach a point at a given depth and distance from the starting point. Although very simple, the example shows almost all the different AUVs behaviors (with the exception of the yaw rotation). Of course, the start and stop points have been chosen so as to avoid a direct, feasible path by one of the AUVs. I.e., all the three AUVs need to have at least one intermediate point. The reference frame is centered on the sea surface with the z-axis pointing downwards.

The start and stop points are:

Start {0,0,0} m at $T_0 = 0$ s
Stop {15,0,40} m at $T_1 = 200$ s – for *Typhoon* and *Remus*
Stop {15,0,40} m at $T_1 = 2000$ s – for *SeaGlider*

for *SeaGlider* the time to reach the stop point has been increased because it is much slower than the other two AUVs.

It is worth noticing that, as we said in the previous sections, the AUVs are supposed to start their movements with the right direction, i.e., the very first rotation is neglected. As a consequence, in order to show a rotation along the yaw axis, it is necessary to have a course of three or more points. Of course, without the yaw rotation, any 3D path can be translated to a 2D one. As a consequence, we decided to use 2D graphs to show the AUVs movements in order to better highlight the different AUVs behaviors.

The AUVs parameters are summarized in Table 1 (they can be changed though Attributes).

Each AUV type has its own limitations. Some parameters have a maximum and minimum value, while some others are marked with the sign '–', which means that the correspondent physical quantity is not considered in the kinematic model. For example, *Typhoon* does not usually modify its pitch angle during a mission, maintaining it around the zero value.

Table 1: List of cinematic and energetic parameters: Typhoon ('Ty'), Remus ('Re') and SeaGlider ('Gl').

Parameters	Ty	Re	Gl
Max. operating depth [m]	300	100	1000
Max. heave speed [m/s]	0.26	–	–
Cruise. surge speed [m/s]	1.5	2.3	0.5
Max. surge speed [m/s]	2.5	2.3	0.5
Max. pitch rate [rad/s]	–	0.3	–
Max. yaw rate [rad/s]	0.3	0.3	0.3
Max. pitch [rad]	–	1.05	1.045
Min. pitch [rad]	–	–	0.24
Power const. heave motion [W s^3/m^3]	1200	–	–
Power const. surge motion [W s^3/m^3]	110	45	–

The simulation results are shown in Table 2 and Figure 7. Table 2 reports the list of waypoints from the *'start point'* to the *'stop point'* along with the action performed by each AUV (e.g., rotation along an axis, movement, etc.).

It can be seen that the proposed mobility models take into account all the constraints described in Section 3. In particular, the rotations and the time required to reach an attitude from a previous one. The roll angle (see Figure 1) is always neglected. Nevertheless, its value is usually fixed at a specific value (zero, in our case), because its change can lead to instability.

From a qualitative point of view, it easy to understand that the mobility model for *Remus* is more complex than the one for *Typhoon* because *Remus* has more choices. As an example, we decided to use a single dive path and one horizontal path, but also a double dive path (a zig-zag path similar to the *SeaGlider* one) could have been performed.

The *SeaGlider* model complexity is similar to the *Remus* one, with the simplification that the path is bound to be a zig-zag one.

Of course, there could have been a number of other equivalent paths. As a matter of fact, any path with a lesser slope (for *Remus* and *SeaGlider*) would have meant more intermediate points, more changes of direction, a longer overall path, and more consumed energy. Only for *Typhoon*, an increase of the intermediate points would have no effect on the energy consumption, but we decided to prefer a simpler movement over a more complex one.

We also derived the total energy consumption for each AUV. *Typhoon* consumed 3419.19 J, *Remus* 10978.53 J, and *SeaGlider* 1.6 J. The difference is striking, but we want to stress that the scenario used is particularly well suited for Glider. Moreover, the energy consumption is not linearly dependent on the speed, and using a maximum speed in some parts of the path is not the right choice.

5 CONCLUSIONS

In this paper, we proposed a new model to take into account the kinematic constraints of an AUV. The proposed system can be easily extended to other kinds of mobile objects, like aerial drones (UAVs) or cars.

The separation of concerns allows a better and faster development of position-dependent algorithms because the algorithm can focus on *where* to move the node, and not on *how* to perform the movement.

Table 2: List of AUVs actions

Time (s)	X (m)	Z (m)	Action
Typhoon			
0	0	0	Dive
153.85	0	40	Go straight
200	15	40	Stop
Remus			
0	0	0	Set pitch angle
3.5	0	0	Go straight
23.55	22.95	40	Set pitch angle
27.05	22.95	40	Set yaw angle
37.52	22.95	40	Go straight
200	15	40	Stop
SeaGlider			
0	0	0	Dive
1000	21.13	36.42	Set yaw angle
1010.47	21.13	36.42	Dive
2000	15	40	Stop

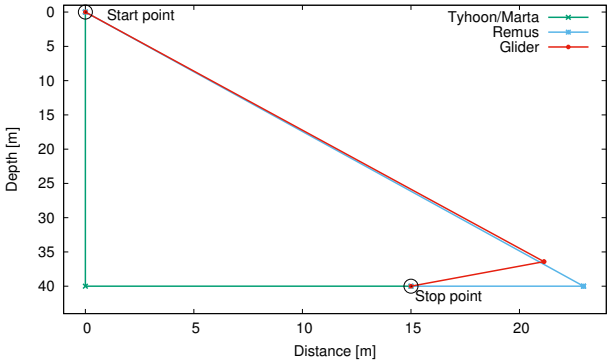

Figure 7: Simple path of three AUVs.

While the new models represent an improvement over the existing ones, several improvements could be foreseen for them.

About the AUV models, in particular, the attitude changes have been extrapolated and should be properly validated. About the energy models, the consumption for some movement types (in particular the yaw rotations) is not yet modeled, and the energy minimization should be considered when choosing a path (in particular for what concerns the object speed).

Another possible refinement is to move the attitude representation in the main Mobility classes, in order to have a standard description of where the mobile object is pointing. With this improvement, the node would be represented by:

- Three coordinates for the position,
- Three angular coordinates for the attitude,

- $n*3$ sets of values for the movement description (dependent on the mobility model).

Moreover, the mobility model could (or could not) modify the node attitude during a movement. As an example, a node could be able to move sideways without changing its attitude. This attitude representation will be needed for UAVs, where the effects of the wind can bot be neglected and the actual node attitude can not be derived from the movement direction.

Another improvement could be to improve the mobility models in order to consider more realistic movement types (e.g., by taking into account the body mass and the time to reach a given speed). Moreover, the composition of different mobility models should be allowed. This could enable the development of disturbance models like marine currents and tides (or wind for aerial drones).

An even more precise representation would be to use the real node dynamic behavior, i.e., to take into account the mechanical and electrical transients, the body mass, body shape, etc. However, this would probably lead to an over-accurate model, not useful in a high-level simulation. For this reason, we believe that the simplified models are more appropriate for ns-3 and that only the marine current (or the wind) effects should be part of a future development.

The antenna radiation (or the acoustic modem directional pattern) can be easily described with proper antenna model extensions. In any case, we believe that the node attitude and the antenna models should be integrated more tightly, in order to allow more precise results, consistent with the actual antenna position on the node (e.g., top/bottom placement).

The proposed kinematic solution has been tested in the simulation phase of the SUNRISE/BRUCE EU project. The goal was to enhance the network connectivity among underwater heterogeneous communicating acoustic nodes, exploiting a mobile AUV that acts as a bridging node. The proposed model greatly simplified the scenario definition, allowing to set only the relevant waypoints in the AUV path.

The code is available at https://codereview.appspot.com/312460043.

ACKNOWLEDGMENTS

This work has been supported by the project SUNRISE/BRUCE, that has received funding from the European Union's Seventh Framework Programme within "SUNRISE - Open Call 2 for new beneficiaries" under Grant Agreement No.: 611449.

REFERENCES

[1] B. Allen, R. Stokey, T. Austin, N. Forrester, R. Goldsborough, M. Purcell, and C. von Alt. 1997. REMUS: A Small, Low Cost AUV; System Description, Field Trials and Performance Results. In *OCEANS '97. MTS/IEEE Conference Proceedings*, Vol. 2. Halifax, Nova Scotia, Canada, 994–1000. DOI: https://doi.org/10.1109/OCEANS.1997.624126

[2] B. Allotta, A. Caiti, R. Costanzi, F. Di Corato, D. Fenucci, N. Monni, L. Pugi, and A. Ridolfi. 2016. Cooperative Navigation of AUVs via Acoustic Communication Networking: Field Experience with the Typhoon Vehicles. *Autonomous Robots* 40, 7 (2016), 1229–1244.

[3] B. Allotta, R. Costanzi, A. Ridolfi, C. Colombo, F. Bellavia, M. Fanfani, F. Pazzaglia, O. Salvetti, D. Moroni, M.A. Pascali, M. Reggiannini, M. Kruusmaa, T. Salumäe, G. Frost, N. Tsiogkas, D.M. Lane, M. Cocco, L. Gualdesi, D. Roig, H.T. Gündogdu, E.I. Tekdemir, M.I.C. Dede, S. Baines, F. Agneto, P. Selvaggio, S. Tusa, S. Zangara, U. Dresen, P. Lätti, T. Saar, and W. Daviddi. 2015. The ARROWS Project: Adapting and Developing Robotics Technologies for Underwater Archaeology. *IFAC-PapersOnLine* 48, 2 (2015), 194 – 199. DOI: https://doi.org/10.1016/j.ifacol.2015.06.032

[4] P. Casari, C. Tapparello, F. Guerra, F. Favaro, I. Calabrese, G. Toso, S. Azad, R. Masiero, and M. Zorzi. 2014. Open Source Suites for Underwater Networking: WOSS and DESERT Underwater. *IEEE Network* 28, 5 (September 2014), 38–46. DOI: https://doi.org/10.1109/MNET.2014.6915438

[5] C. C. Eriksen, T. J. Osse, R. D. Light, T. Wen, T. W. Lehman, P. L. Sabin, J. W. Ballard, and A. M. Chiodi. 2001. Seaglider: A Long-Range Autonomous Underwater Vehicle for Oceanographic Research. *IEEE Journal of Oceanic Engineering* 26, 4 (Oct 2001), 424–436. DOI: https://doi.org/10.1109/48.972073

[6] F. Guerra. 2017. WOSS - World Ocean Simulation System. (2017). http://telecom.dei.unipd.it/ns/woss/

[7] J. Potter, J. Alves, D. Green, G. Zappa, I. Nissen, and K. McCoy. 2014. The JANUS Underwater Communications Standard. In *2014 Underwater Communications and Networking (UComms)*. Sestri Levante, Italy, 1–4. DOI: https://doi.org/10.1109/UComms.2014.7017134