Simulator for Naiad [*]

Daniel Lindqvist
dlt09003@student.mdh.se

ABSTRACT
Testing and monitoring the AUV are two of the key features of the simulator. When the simulator is connected to the AUV all the information that transferred over the CAN bus between different subsystems will be sent to the simulator. This information can either be used for monitoring the AUV if all sub systems are running together with the option of simulating a certain device by sending out outputs just as if it was the real device doing so.

The implementation of the Simulator was written in Ada with an object oriented approach following the Model View ViewModel standard described in [3]. The Model View ViewModel approach puts all the logic away from the User Interface layer of the code and into the ViewModel layer used only for GUI logic. This allows for proper testing for all more complex parts of the code.

The formulas used for simulating the movement of the AUV are based on numerical integration of the accelerations. The accelerations are calculated using the the general space manipulation formulas with simplified frictions assuming symmetry of the AUV which was found feasible in [2].

The backbone and the motion simulations of the simulator have been implemented and tested together with the motion control both as function calls and with CAN messages over Ethernet to test and validate the realistic motion together with the correct data transfers and responses. This yielded realistic movement patterns both with regards to the positioning as well as the orientation.

1. INTRODUCTION
The simulator is a test platform for the AUV, which is connected to the AUV with full access to all information sent between the different sub systems of the AUV. The task of the simulator is that it should read all data sent from the AUV in order to either observe its movement or use the data for simulating a sub system of the AUV by sending out data over the CAN bus. This allows the simulator to see the response of the other parts of the system, such as the wanted motor power and which actuators that would be used. The calls to the actuators are then being observed in order to make the same actions in the simulator to feed the AUV with simulated sensor data.

[*]This report was written during the fall of 2013 in an advanced level project course at Mälardalen University, Sweden.

1.1 Observation
When the simulator is observing the AUV regarding how it behaves in water with regards to position orientation. At this point the data will not only be showed on the GUI but will also be used as vital information that can be used in the future for improve the simulator's performance.

1.2 Simulation at a pace faster than real-time
The final task of the simulator is to be a platform for testing the motion control algorithms in a simulated pace which is faster than real-time in order for AI based approaches to be able to evolve within reasonable time frames. Either for tuning the settings of the motion control or for creating AI based algorithms for the motion control.

1.3 Representation
The presentation of the simulator will be shown with the position in three planes, which will make it easy for the user to track the movement patterns of the AUV. To compliment this a three dimensional view of its orientation will give the user a clean and easy to understand representation of how the AUV moves.

2. METHOD
Approaches used for solving problematic areas regarding the the implementation of the simulator, such a the overall structure of the program, simulated motion, representation and communication is explained in this section.

2.1 Representation
To represent the simulator, the choice of using GTK was obvious since the simulator was written in Ada. The reason for this was documentation, even though even though it was not greatly documented it was still far better than any of the other options that almost completely lacked documentation. The aspect of documentation was also combined with all

the graphical tools that were needed to create the clean, informative and user friendly interface. Another aspect for making GTK Ada the better choice was that it was available on multiple platforms including Windows and Ubuntu.

2.2 Design

For the structure of the program between the Graphical interface, the motion calculation and the Ethernet sockets for communication with the AUV, the Model View ViewModel (MVVM) approach was decided upon. This is due to the clean coding with a clear interface between each layer in the program and a simple way of integrating the pieces. Here the top part which is the graphical interface calls for the underlying layers. This follows both the object oriented approach that is used for this project and implements well with Ada AUnit testing as well as there will not be any logic mixed up with the interface layer using that approach making parts easier to test.

2.3 Motion Simulation

Simulating the motion of the AUV is based on numeric integration of the accelerations that can be calculated using the functions described below. The method that will be used is to calculate the accelerations in a local coordinate system. The features such as the form of the AUV, motors and torques are already defined in the local coordinate system. This results in converting the velocities and angular velocities to the local frame using the inverse of the orientation matrix used to describe the orientation of the AUV. The local velocities are then used for the acceleration formulas as it affects both the friction force and the angular acceleration straight of, using formulas for general space manipulation described in the book Robotics, Vision and Control [1].

$$GlobalVector = Orientation * LocalVector \tag{1}$$

$$Acceleration = \frac{\sum Force}{Mass} \tag{2}$$

$$J * \alpha = -\omega \times J \cdot \omega + T \tag{3}$$

The moment of inertia and the mass of the AUV are both automatically calculated in the Solid Works CAD program used for designing the hull of the AUV. The affect of friction is less predictable and hard to mathematically calculate without expensive software. Assumptions regarding that a close to symmetric object actually is symmetric will be used to simplify the problem while still resulting in accurate approximations according to [2].

The accelerations are then converted back to the Global reference frame, allowing for the numerical integration of the velocities, positions, angular velocities and orientation. This results in the orientation being represented as a matrix and the other sets of data as vectors.

2.4 Communication

The communication of the simulator will be used to see the flow of data going between the different parts of the AUV, which is passed on by the sensor fusion module to the simulator. The simulator will send out messages when in simulation mode, where it simulates a sub system. This is done by receiving all inputs and sends corresponding outputs for that specific subsystem.

The communication between the simulator and the BeagleBone Black, which is what the simulator will be connected to in the AUV will be done over Ethernet. For communicating over Ethernet TCP sockets will be used as they are easy to use and give reliable performance. The set of data sent over the TCP will be following the same standard as the data sent over the CAN bus, which will make it simple to forward data by just pushing all data from CAN to Ethernet and vice versa without having to convert the messages.

3. IMPLEMENTATION

The implementation of the simulator for project Naiad was created with the four following pillars in mind and tested with AUnit to ensure that all parts works as intended.

3.1 Overall design using the Model View ViewModel approach

The overall design of the simulator was implemented following the MVVM standard. At the top of this design is the graphical interface for the user. The graphical interface in turn holds several view models, which it gets its data from and calls on for updates of underlying layers. Each view model containing its own logic separated to that of the model itself to perform simpler tasks that is just for representation. There are 7 view models that all work on different parts of the program such as one for graphical representation, one for the PID values, one for the PID errors and one for setting each type of parameter.

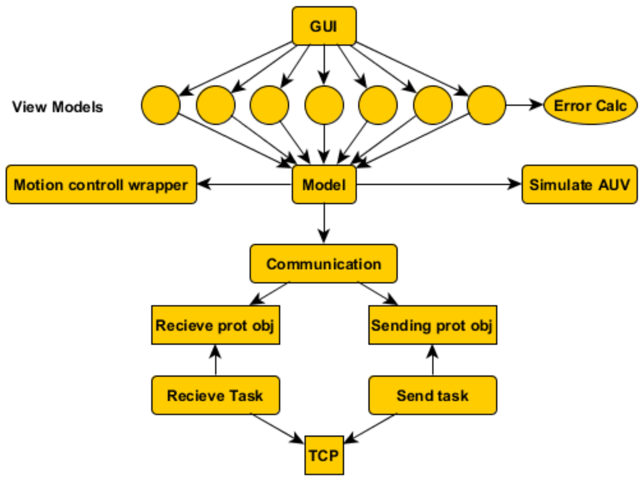

Figure 1: Simulator user interface

These view models then shares one common model that they are all keeping access to. This model in turns holds some simple logic together with the motion simulator part of the simulator and the different communication accesses, both with CAN over Ethernet and with a direct link to motion control. Where it with the update command fetches information of the simulated motion part and the communication part and sends it to the other respective part.

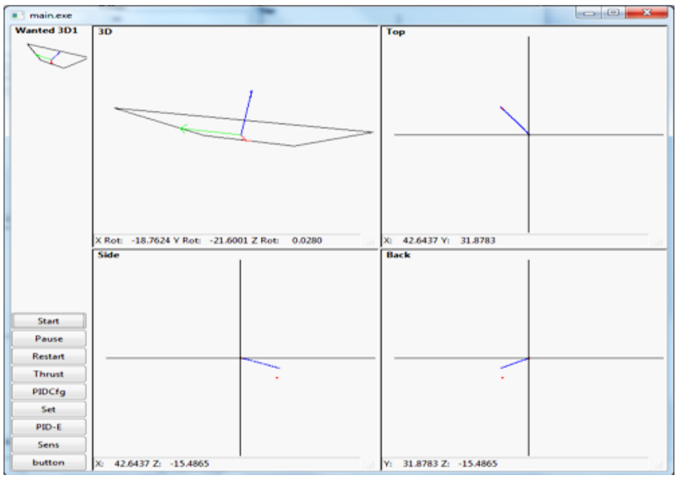

Figure 2: Simulator user interface

3.2 Usability and design in GTK Ada

The design of the user interface is built up as in Figure 1 where the three views showing positioning in the two respective axis.

The interface is built up from using buttons where popup windows appear for setting data or monitoring actuators such as the motors without bloating the screen. To perform actions with a logic free interface the accessible view models are used to perform the various actions such as setting values, retrieving information, calling the periodic update for the simulated AUV and so on.

3.3 Motion Simulation

The implementation of the motion simulation is based on calculating the torques and forces from the motors. These are then used together with the inertia, frictions, buoyancy and other aspects stored in the simulated motion to calculate the acceleration and angular acceleration that they would affect the AUV with on that specific time step.

The acceleration and angular acceleration is then used with numerical integration to receive the changes in velocity and angular velocity during that time frame, combining it with previous knowledge giving velocity and angular velocity. The velocity and angular velocity are then used in the same way and integrated into position and orientation.

3.4 Communication

The design for the communication over TCP consists of two tasks and a interface that is used for communicating with those two tasks. The sender task retrieves data from a protected object, a set of information that could safely be used by different threads. From this protected object a sender task retrieves the data from the protected object that it is required to create the respective CAN messages to send.

On the other side of the communication package a receiver task reads the incoming CAN messages and writes the informative data into a receivers protected object which it is read when the simulator requests data from the communication section.

3.5 Unit tests with AUnit

For testing the sub parts of the system AUnit was used to test that each piece of logic was working as expected in order to fulfill the overall performance requirements of the system. All the possible correct outcomes were to be predicted so that each sub system could be guaranteed to work properly. Though in the end as time was a key issue it was implemented on all key parts of the system which was easily accessible for testing and had a quite high risk of failure. Priorities to not test all part of the simulator was done where risks of failure was not high as the functionality was simple and could be tested lightly in other ways. This ended up saving key time while still getting the overall results that were wanted from the simulator in time.

4. RESULT

The latest version of the simulator by January 2014 the simulator reached feasible results, reaching up to the three main objectives which were observing, simulation and communication over Ethernet. With all data being represented in a informative way.

4.1 Representation

The graphical interface used to represent the data from the simulator in a clean, informative and user friendly way ended up meeting those requirements. The interface shown in figure 1 is demonstrating how clean and easily accessible the data is with buttons for popup menus are used for setting other data points.

4.2 Simulated Motion

The validation of the AUV's movement in water is heavily restricted without testing the real AUV in water. This is since the movements of the AUV are hard to predict forehand for creating unit tests properly, especially wanting to validate the results without using the same or similar methods as used to create the simulator itself. Some basic methods were used to check for miss behaviour of the system in both planar and rotational movement.

4.2.1 Translational movement

The motion of the simulator based on different forces works as expected when it comes to planar movement without any rotations. As the motors' forces are set to counteract gravity and then pushing the AUV in a certain direction gives the expected axis of accelerations with realistic ratios between them.

4.2.2 Rotational movement

The movement in the orientation view of the simulator is less easy to predict regarding how the AUV should act, the buoyancy force together with friction always manages to stabilize the z-axis of the AUV with the z-axis of the reference frame when motors are turned off, which is a reasonably good sign but far from a proof that it works as intended.

With the motors set to create a torque and no planar acceleration, resulting in the AUV being in the same position while changing orientation. With the change in orientation being around an axis that is close to that of the combined torque, which was also as expected as the inertia will make make the rotation axis slightly off.

4.3 Communication

The communication over Ethernet using TCP was confirmed to be working after tests of including sending different CAN messages over TCP that the simulator was supposed to receive and the printed out the data set of received data. Using the TCP the information is already confirmed to be intact as long as the data sent was not corrupted before sending. This results in a stable connection with the AUV. During the tests the communication was working as intended with regards to both sending and receiving.

5. CONCLUSION

The results that were achieved with the simulator lived up to the main expectations that it was set up to achieve. Where the main part of the simulator was to work as a test platform for the motion control as well as observing the AUV in the water with regards to position and orientation.

This allowed the simulator to be used both as a check for the motion control but also will allow the simulator to observe the AUV. Even though the simulator is not fully realistic with regards to the approximations of frictions and inertias the approximations were accurate enough for testing a self balancing system such as the PID controlled motion control. It would require new tuning values for the PID controllers to optimize overshoot and stabilizing times the AUV when in water.

The communication lived up to expectation by getting all the information across from point A to point B. Tests were done to ensure that all relevant CAN messages were sent and received according to the standard set-up for the CAN communication as can messages are sent over Ethernets TCP sockets.

5.1 Future work

Future work would involve increasing the modularity and the systems that could be simulated in the simulator together with choices about which sub systems that were to be simulated. Where each sub system would be running on its own with its own communication between other simulated sub systems and also one for external systems in the real AUV

Another area of further development would be to improve the approximations of movement for the simulated motion of the AUV. This could and should be done using logged data about how the AUV moves in the water depending on different parameters. Improvements could be implemented either through constant tuning AI or creating algorithms based completely on AI in order to try and approximate the reality in a more precise way.

6. REFERENCES

[1] P. Corke. *Robotics, Vision and Control Fundamental algorithms in matlab®*. Springer, 2013.

[2] C. Jian-Hua, W. Xin-Zhe, C. Xu-hong, and H. Yan-ling. Research and design of pins simulator based on underwater vehicle space model. In *Knowledge Discovery and Data Mining, 2009. WKDD 2009. Second International Workshop on*, pages 917–920, 2009.

[3] Wpf apps with the model-view-viewmodel design pattern. `http://msdn.microsoft.com/en-us/magazine/dd419663.aspx`. Accessed 2014-01-13.