How logfiles are read in Travis

Kristof Kipp
University of Bremen
Bremen, Germany
kkipp@informatik.uni-bremen.de

Nils Leusmann
University of Bremen
Bremen, Germany
leusmann@informatik.uni-bremen.de

Ingmar Ludwig
University of Bremen
Bremen, Germany
iludwig@informatik.uni-bremen.de

***Abstract*—This paper introduces three layers of communication from the reading of log data that was written by an AUV during a mission to the simulation provided by the Travis project team. The first layer handles data reading and providing of raw data in primitive c++ data types. A second layer prepares these data and provides the means for the Unreal Engine to read the data. The third and highest layer reads the prepared data of the second layer and uses it to manipulate the simulated model within the Unreal Engine.**

***Index Terms*—unreal engine, unreal, trajectory, Travis, visualisation**

I. Introduction

After talking about how to handle the data we got from *MARUM* (which is disclosed from public), we decided to create a multi-layer architecture for handling and presenting the data. This decision was made due to the sheer amount of data (approx. 500k lines per logfile) and the possible priotization of data importance (e.g., some of the data might be neglectable due to physical reasons). Every layer has its own chain of responsibility and needs to be described to a certain degree. The interaction between the layers is done via layer-specific calls (e.g., via *UPROPERTY*).

II. Data Layer

This layer is responsible for the reading of the log file and efficient storage of all data concerning the log file data.

A. Logfiles

The log files are available in pure CSV format and contain all data logged on a mission, sample files provided by MARUM contain about 500000 lines, so redundand data holding is highly discouraged. Therefore we propose a single layer for holding and maintaining the data. The set of sample data contains two files: *log_20160516_dive.csv* and *log_20160517_dive.csv*. They both contain a large set of data columns of which only a subset is actually needed for the means of this project. The complete set of colums can be found in section 5 of this paper. Despite it being a huge chunk of columns this layer will only handle I/O operations concerning the data.

B. Reading of the Log file

The log file itself is a huge raw text file in the csv file format[1] consisting of data logged by any of the sensors, the file is comma-seperated and delimited by either of the CR (carriage return) and the LF (line feed) control character.

The reading process starts by storing each line into a string vector. This vector will increase in size and will hold all the information in an unsorted way. For further processing purpose we needed to make the data easier accessible. Hence we created an data object which would store all the data of one column. Here each column, which describes one data value, will get its own variable according name. This data type is called *logdata_t*. To have all lines accessible in one variable we created an *logdata_t* vector, which will hold all the *logdata_t* objects for each row. The MARUM assured us that the order of the sensor data will always be the same, because of this concession we created an function which separates the line strings at a comma and then stores the values of each sup-string according to their position to the variables. To increase the performance we will be using multiple threads while converting the huge string vector into a *logdata_t* object. This means we will have multiple threads converting many different lines at once. Because of the thread pool library we are using, it is only possible to give a thread one parameter. For the thread to work properly it needs more information. This means we needed to create an new data object which will be passed to each thread. The *logline* consists of the variables seen in table I. Because the C++ std::vector is not thread safe we created an mutex lock for the data vector.

C. Functions on Layer 1

- **‘public int readLog(string logfile)‘**: opens the file that is passed in the argument ‘logfile‘ and reads its contents.
- **‘public logdata_t[] GetAllData()‘**: returns the complete data set (most likely used for building the initial trajectory in the simulation.
- **‘logdata_t GetDate(float time)‘**: returns a dataset by the given timestamp (see section 5, first column of csv file)
- **‘logdata_t GetDate(int idx)‘**: returns a dataset by the given index (e.g., data[i])
- **‘logdata_t *GetDateGeqTime(double time)‘**: Gets the first date where the logdata_t.time¿time
- **‘logdata_t *GetDateLeqTime(double time)‘**: Gets the first date where the logdata_t.time¡=time

[1] https://tools.ietf.org/html/rfc4180

Human readable String	Description
int idx	The row in which the data will be stored (needs to be -6)
std::string line	The actual data from the line
ascdata_t* _data	A pointer to the dataobject in which the data will be stored
AquariumHelper* _helper	The aquarium in which the points should be converted

TABLE I
THE MEMBERVARIABLES OF THE LOGLINE

III. COMMUNICATION LAYER

The communication layer does not store any of the raw data. It will only convert the needed Data (and only when it is needed) from the Data Layer into an format that can be used in the presentation layer (Unreal Engine). The difficulty here is that the presentation layer will use Unreal Engine Blueprints, which is limited in some ways. For example the presentation layer can not work with *double* values[2]. In addition the presentation layer is incapable of using polymorph functions[3]. Next to some downsides the Unreal Engine entails it also provides some unique opportunities like special Datatypes, which will be very useful later on, for example the *FVector*.

A. Design

The second layer will be represented in the Unreal Engine as an Unreal *Aactor*-Object. This Object is called the DataCommunicator. The main reason for this representation is the fact, that we wanted the DataCommunicator to be easily accessible in the Levelblueprint. In addition to this we want the Object which represents the communication layer to be single point of interest in the Unreal Engine. In other words we want the Presentation Layer to always turn to the DataCommunicator if it wants to know something about the AUV. A positive side effect of the fact that the Communication Layer is represented as an Aactor-Object in the Unreal Engine is, the fact that we can track the time since the start of the simulation via the Tick() function.
The Communication Layer will only store two different Data points, in an Blueprint accessible format, at any time. The values of these these Data points will immediately translated into a blueprints and needs to be changed according to the wishes of the Presentation Layer. The Presentation Layer will only need two different Data Points at any time. The reason for two data points is due the fact, that the whole Data set is a discrete set which we try to represent in a continuous way (time). If the simulation time is between two data points the presentation layer will interpolate the value it wants to show. All functions which update the available Data in the Presentation Layer will signal if the updating was successful (true) or not (false).

B. Communication

The Data Layer will be initialized in the Communication Layer, so the Communication Layer always has a direct connection to the Data Layer which allows it to access the read data. As already mentioned in III-A the Communication Layer stores only two data points at any time. This is technically correct, but it also stores two pointer to the corresponding data points in the Data Layer. So it can always access the currently active data points in both layers. Once as it is in the Data Layer, and once as it will be made available to the Presentation Layer. The difference between these two data objects (*log-data_t*,*UWaypoint*) is the precision. The communication with the Presentaion Layer will be done via *UWaypoint* objects. The *UWaypoint* class is a class specially created for this purpose. It inherits from the *UObject*-class from Unreal, which means we can reference to it inside the Engine. Inside the class all important data values for one specific time stamp. Each data point inside the Waypoint is a variables which is made accessible for the Blueprints via usage of *UPROTPERTY*. Usually the variables should be *BlueprintReadOnly* because we do not want the Presentation Layer to be able to change the Data. The Objects of this class which will be created by the DataCommunicator and also need to mad accessible via the *UPROPERTY(BlueprintReadOnly)*. To change the data values which are currently available to presentation layer the communication layer offers specific functions, which will updated the stored values accordingly.

C. Procedure

The Task of the Communication Layer is to convert the raw C++ Data into Datatypes which are accessible in the Unreal Engine. For this Task the DataCommunicator is the connection between both sides. At the beginning of the simulation the DataCommunicator will be initialized via the *StartCommunication()* function. This will and needs to be called in the Levelblueprint right after the beginning of the game. Now the DataCommunicator will read the Travis Configuration file. The configuration file is editable via an launcher and stores the absolute file path to the .log file (log of the mission), the .asc file (height map data) and an possible shift and rotation for the height map. After reading the configuration file the Communication Layer will begin to initialize the Data Layer Object. This Object will need a Path to an Mission-log file and begin to read it. Afterwards the DataCommunicator has a direct connection to the Data Layer and access the values. Subsequently the aquarium will be created. Followed by the creation of the bathymetry, if there is already an corresponding Travissave-file for the chosen log file the DataCommuicator will read the save file and create the map accordingly. If not then it will create an height map accordingly to the data stored

[2]https://answers.unrealengine.com/questions/89591/blueprint-does-not-have-double-type.html
[3]https://answers.unrealengine.com/questions/553278/inheritancepolymorphismoverriding.html

in the .asc file. Finally after the initialization phase it will give control over to the Presentation Layer.

D. Functions of Layer 2

The Communication Layer provides the following functions:

- **UFUNCTION(BlueprintCallable, Category = "Travis") bool StartCommunication()**: Initializes the Communication Layer and makes the first Points accessible
- **UFUNCTION(BlueprintCallable, Category = "Travis") void SetHeightMapMaterial(UMaterialInterface* material)**: Sets the Material of the HeightMap
- **UFUNCTION(BlueprintCallable, Category = "Travis") FVector GetRandomPointonHeightMapInAquarium()**: Gives an Random Point on the Heightmap inside the Aquarium
- **UFUNCTION(BlueprintCallable, Category = "Travis") FVector GetMiddleOfAquarium()**: Returns the middle of the Aquarium
- **UFUNCTION(BlueprintCallable, Category = "Travis") bool getNextPointFromIndex(int dataindex)**: Gets the point of data from index and make it accessible to the presentation layer
- **UFUNCTION(BlueprintCallable, Category = "Travis") bool getNextPoint()**: gets the next Datapoints for the current time and make it accessible to the presentation layer
- **UFUNCTION(BlueprintCallable, Category = "Travis") void SetWarp(float multiplier)**: changes the timewarp
- **UFUNCTION(BlueprintCallable, Category = "Travis") float GetWarp()**: gets the timewarp
- **UFUNCTION(BlueprintCallable, Category = "Travis") void SetTime(float time)**: Sets the simulation to the current time
- **UFUNCTION(BlueprintCallable, Category = "Travis") void ResteTimeToMissionStart()**: Resets the time to the Mission start

The most interesting function is StartCommunication(), which is already described in III-C.
To Get an random Point on the HeightMap, we search for a random Point in the aquarium and then make an raycast downwards and return the position where we hit something. To make sure we only hit the heightmap and no other Object in the Level, we created an own Collision Channel in the default setting are ignore. Only the Heightmap is set to block Raycasts on this channel so we are sure that, if we hit something we hit the Heightmap.

IV. Presentation Layer

The presentation layer is responsible for three tasks: Acquiring the right data from the communication layer (depending on the user settings and the current simulated time), preparing the received data and transferring it to the visualizations.

The acquiring of the data is done using the excess point of the communication layer and several methods for changing which data is received. The access point consist of the (invisible, but in the level physical present) DataCommunicator Object which can be used to access all the needed data. This is done by getting a reference to the DataCommunicator object and using its getter methods, which is initialized using its Start Communication method.

The DataCommunicator Object directly offers several getters from which the three most important are Get BP Waypoint 0, Get BP Waypoint 1 and Get Alpha.

The two getters Get BP Waypoint 0 and get BP Waypoint 1 each return a so called Waypoint-object. A Waypoint-object represents one point in time at which the AUV has written data to the log file. Waypoint 0 is the closest Waypoint in time for that applies $t_{WP0} <= t_S$ with t_{WP0} being the time the waypoint was recorded and t_S being the current simulated time. Analogous Waypoint 1 is the closest Waypoint in time for that applies $t_{WP1} > t_S$.

A Waypoint-object offers several getter methods for AUV status data, e.g. the AUVs Speed at the specific Waypoint.

Get Alpha returns the interpolation factor for the two Waypoints, calculated using the current simulation time. Using this value the data for every point between the two waypoints can be interpolated.

Before the data is accessible through the Waypoint-objects and the Get Alpha method the right data has to be set by the communication layer. This is requested in the level blueprint through using one of two methods: Get Next Point From Index or Get Next Point. Get Next Point from Index sets the Waypoints using their index (e.g. Get next Point From Index(0) returns the first and the second Waypoint). This method is used for drawing the trajectory. The second method Get Next Point sets the Waypoints and the Alpha-value according to the current simulation time. This method requires a call of the Reset Time To Mission Start method before it can be used the first time and after every use of the Get next Point From Index method.

The received data is then processed depending on the visualization and the data. For fast changing visualizations with easy visible changes (e.g. the AUV movement) the values of the two waypoints are interpolated using Unreals own linear interpolation algorithm. For slow changing visualization (e.g. the pressure visualization) this is not necessary due to the large number of Waypoints. For the purpose of optimizing the computation time the value of the first Waypoint is passed to the visualization while the two Waypoints remain the same. Without interpolation this leads to approximately 10 updates of the value per second (vs. one update per tick while using interpolation). For an Overview over the executed modification Refer To Table I, II and II.

While using the Get Next Point method to receive data, the simulation can be influenced using one of two methods: First, the simulation time can be set using the Set Time method, e.g. to jump to the middle of the simulation. Second, the speed of the time laps can be increased or decreased using the Set Warp

method, which takes a factor that is multiplied with the speed of the time laps (e.g. 1 for normal time laps and 2 for double speed).

All the used Values obtained from the Waypoints and the DataCommunicator getters are further described in Table I, II and II.

V. Appendix: Logfile Layout

- vcc_clock_real_seconds
- vcc_log_counter
- vcc_dvl_altitude_fb_m
- vcc_altimeter_altitude_fb_m
- vcc_em2040_altitude_fb_m
- vcc_dvl_altitude_valid
- vcc_em2040_status_word
- vcc_battery_fb_volts
- vcc_battery_current_fb_amps
- vcc_energy_used_kwhs
- vcc_energy_used_percent
- vcc_thruster_current_fb_amps
- vcc_man_heading_sp_deg
- vcc_dgps_longitude_fb
- vcc_dgps_latitude_fb
- vcc_dgps_quality_fb
- vcc_phins_longitude_fb_deg
- vcc_phins_latitude_fb_deg
- vcc_man_pitch_sp_deg
- vcc_phins_pitch_fb_deg
- vcc_plane_1_sp_deg
- vcc_plane_1_fb_deg
- vcc_plane_2_sp_deg
- vcc_plane_2_fb_deg
- vcc_plane_3_sp_deg
- vcc_plane_3_fb_deg
- vcc_plane_4_sp_deg
- vcc_plane_4_fb_deg
- vcc_plane_5_sp_deg
- vcc_plane_5_fb_deg
- vcc_man_roll_sp_deg
- vcc_phins_roll_fb_deg
- vcc_thruster_rpm_sp
- vcc_thruster_rpm_fb
- vcc_thruster_rpm_sp_profiled
- vcc_dgps_speed_fb_mps
- vcc_system_mode_fb
- vcc_ctd_temp_fb_degc
- vcc_hull_temperature_fb_degc
- vcc_dvl_temperature_fb_degc
- vcc_phins_heading_fb_deg
- vcc_speed_sp_mps
- vcc_speed_fb_mps
- vcc_thruster_modelled_speed_fb_mps
- vcc_man_depth_sp_m
- vcc_pos_depth_fb_m
- vcc_man_altitude_sp_m_actual
- vcc_pos_altitude_fb_m
- vcc_man_altitude_depth_sp_m
- vcc_man_vert_control_mode
- vcc_man_depth_control_mode
- vcc_mission_line_heading
- vcc_mission_line_offline_distance
- vcc_mission_line_output_control_heading
- vcc_man_pitch_force_clipped
- vcc_man_roll_force_clipped
- vcc_man_yaw_force_clipped
- vcc_man_depth_force_clipped
- vcc_thruster_volts_cmd
- vcc_ctd_conductivity_fb_spm
- vcc_ctd_press_fb_dbars
- vcc_ctd_salinity_fb_psu
- vcc_ctd_soundvel_fb_mps
- vcc_ctd_status_ok
- vcc_battery_can_fb_volts
- vcc_battery_can_load_current_fb_amp
- vcc_battery_can_id_fb
- vcc_bottom_avoid_active
- vcc_phins_long_speed_fb_mps
- vcc_phins_trans_speed_fb_mps
- vcc_phins_vert_speed_fb_mps
- vcc_phins_output_status
- vcc_wa_fault
- vcc_gf_gfm3_fault
- vcc_plane_fault
- vcc_thruster_status_alarm
- vcc_battery_timeout_alarm
- vcc_paro_press_fb_psi
- vcc_acsa_hh
- vcc_acsa_mm
- vcc_acsa_ss
- vcc_dgps_utc_seconds
- vcc_acsa_error
- vcc_acsa_mode
- vcc_pos_pitch_rate_fb_dps
- vcc_pos_roll_rate_fb_dps
- vcc_pos_heading_rate_fb_dps
- vcc_phins_error
- vcc_pos_heave_m
- vcc_dvl_bottom_vel_x_fb_mps
- vcc_dvl_bottom_vel_y_fb_mps
- vcc_dvl_bottom_vel_z_fb_mps
- vcc_dvl_bottom_range_b1_fb_m
- vcc_dvl_bottom_range_b2_fb_m
- vcc_dvl_bottom_range_b3_fb_m
- vcc_dvl_bottom_range_b4_fb_m
- vcc_em2040_ping_number
- vcc_altimeter_altitude_fb_m_raw
- vcc_dvl_water_vel_valid
- Time in English

Name in Blueprint	Origin of data in log file	Description	Modifications of data	Destination of Data
relativetime	derived from vcc_clock_real_seconds	Time at which the values were recorded (in seconds since mission-start).	No modifications	User interface (update time slider), Set Time method of DataCommunicator (for time jumps with fixed length)
Vcc Pos Altitude Fb M	vcc_pos_altitude_fb_m	Height over ground as recorded by the AUV. Note: There are also other altitude values in the log-file which do not differ much in our samples. According to the mission these might be better suited.	Only WP0 used	user interface (update altitude display)
Position	derived from vcc_phins_longitude_fb_deg, vcc_phins_latitude_fb_deg and vcc_pos_depth_fb	The Position of the AUV in the Aquarium in Cartesian coordinates (refer the Aquarium Paper for details on the calculation)	Interpolation with Alpha value	Set AUV position (using the move AUV method)
Rotator	derived from vcc_phins_roll_fb_deg, vcc_phins_pitch_fb_deg and vcc_phins_heading_fb_deg	Rotation state of the AUV (heading angle, pitch and roll)	Interpolation with Alpha value (for setting of AUV rotation state), using only WP0 (for display in user interface)	Move AUV method (update AUV rotation state), Update Rotation Visualization method, user interface (display of values)
Vcc Ctd Press Fb Dbars	vcc_ctd_press_fb_dbars	Pressure on the AUV recorded by the CDT device.	Only WP0 used	Update Pressure Visualization method, user interface (display pressure value), Calculatate Color From Values method (visualize pressure on trajectory
Vcc Thruster Rpm Fb	Vcc_thruster_rpm_fb	Rotations per minute of the AUVs propeller.	Interpolation with Alpha value (visualization of Propeller rotation), Only WP0 used (display in Interface, visualize on trajectory)	Update Motor Rpm Visualization method , user interface (display motor rpm value), Calculatate Color From Values method (visualize motor rpm on trajectory)
Vcc Speed Fb Mph	vcc_speed_fb_mph	Current speed of the AUV relative to the water. Note: The Speed relative to the ground might differ from the speed relative to the water.	Only WP0 is used	Update Speed Visualization method (3D Visualization), user interface (display speed value), Calculatate Color From Values method (visualize speed on trajectory)
Vcc Plane 1 Fb Deg	Vcc_Plane_1_Fb_Deg	Rotation state of the AUVs front left hydroplane relative to the neutral position.	Only WP0 used	Move Front Left Plane method (update Hydroplane position), Unreals Make Rotator method (Update hydroplane rotation visualization)
Vcc Plane 2 Fb Deg	Vcc_Plane_2_Fb_Deg	Rotation state of the AUVs front right hydroplane relative to the neutral position. Note: The value is multiplied with -1 for correct visualization because of the 180° difference between the hydroplanes on the right and the left side and the different rotation definition in Unreal and the log.	Only WP0 used	Move Front Right Plane method (update Hydroplane position), Unreals Make Rotator method (Update hydroplane rotation visualization)

TABLE II
DATA MODIFICATION AND TRANSMISSION IN THE PRESENTATION LAYER, PART I

Name in Blueprint	Origin of data in log file	Description	Modifications of data	Destination of Data
Vcc Plane 3 Fb Deg	Vcc_Plane_3_Fb_Deg	Rotation state of the AUVs top hydroplane (or tower) relative to the neutral position. The value is multiplied with -1 for correct visualization because of the different rotation definition in Unreal and the log.	Only WP0 used	Move Tower Submarine method (update Hydroplane position), Unreals Make Rotator method (Update hydroplane rotation visualization)
Vcc Plane 4 Fb Deg	Vcc_Plane_4_Fb_Deg	Rotation state of the AUVs back left hydroplane relative to the neutral position.	Only WP0 used	Move Back Left Plane method (update Hydroplane position), Unreals Make Rotator method (Update hydroplane rotation visualization)
Vcc Plane 5 Fb Deg	Vcc_Plane_5_Fb_Deg	Rotation state of the AUVs back right hydroplane relative to the neutral position. Note: The value is multiplied with -1 for correct visualization because of the 180° difference between the hydroplanes on the right and the left side and the different rotation definition in Unreal and the log.	Only WP0 used	Move Back Right Plane method (update Hydroplane position), Unreals Make Rotator method (Update hydroplane rotation visualization)
Vcc Energy Used Percent	Vcc_Energy_Used_Percent	Percentage of the AUVs Battery that has been already used.	Only WP0 used	Update Auv Interface Values (display used energy)
Vcc Ctd Temp Fb Degc	vcc_ctd_Temp_fb_degc	Temperature in the AUV as measured by the AUVs CDT device.	Only WP0 used	Update Auv Interface Values (display current Temperature)
Time in English	Time in English	Time of the recording of the Waypoint as a string. The timezone is GMT and the string is formatted to the standard English time format.	Not modified	User interface (display simulated time).

TABLE III
DATA MODIFICATION AND TRANSMISSION IN THE PRESENTATION LAYER, PART II

Getter Name in Blueprint	Description of return value	Modifications of data	Destination of Data
Get AUV Size Factor	A factor that represents how the AUV size has to be adapted to approximately match the size of the represented earth surface (for the smallest of the three selectable AUV sizes, for more information over the relationship between the size of the world in the simulation and the represented earth surface refer to the Aquarium paper)	The Value is multiplied with a factor connecting the AUV size with the size of the AUV model uasset in Unreal.	Used to determin the value of Initial Scale AUV (a factor that adjusts the size of the AUV)
Get Middle Of Aquarium	A point that lies in the middle of the Aquarium (for details on the Aquarium refer the Aquarium paper)	Not currently used	Not currently used
Get Random Point on Hight Map	Returns a Random Point on The Map generated from the AUVs Sonar Data.	Not modified	Spawn Actor method (in the Spawn Decoration method which adds decorative Elements like stones to the map).
Get Warp	Current time expiration acceleration factor	The factor is doubled or halved	Set Warp (Used to increase or decrease the time expiration rate depending on the current value).
Get Alpha	Interpolation value for interpolating the values between two recorded datasets. Calculated from current simulation time	No modifications	Unreals lerp interpolation method, used with all interpolated values
Get BP Waypoint 0	Returns an object that offers access to data of the nearest Waypoint with $t_{WP0} <= t_S$. Refer section presentation level.	Object is not modified	Very frequently used in the whole level Blueprint
Get BP Waypoint 1	Returns an object that offers access to data of the nearest Waypoint with $t_{WP1} > t_S$. Refer section presentation level.	Object is not modified	Very frequently used in the whole level Blueprint

TABLE IV

DATACOMMUNICATOR GETTER METHODS, MODIFICATION OF RETURN VALUES AND TRANSMISSION OF DATA