Self-supervised Mobility Assessment from Unsupervised Proprioceptive Feature Learning on Simulated Environment

Ian Hughes, Jose Luis Verdugo, Andres Cárcamo, Innervycs

Eric Mark, Jose Miguel Larenas, US Army DEVCOM

Paramsothy Jayakumar, US Army DEVCOM, Ground Vehicle Systems Center

Feb. 2022

Abstract—This work leverages open source tools to create a virtual environment and vehicle that compare to real world conditions in order to tests, train and evaluate different machine learning approaches. In particular, the combination of a Convolutional and aLong Short-Term Memory Recurrent Neural Networks under a Self-supervised learning strategy to avoid extensive labeling and enable the learning of new patterns and terrain, particular that affect or restrict mobility . In the self-supervised structure, the LSTM produces continuos results to provide to the CNN. Since the camera is front looking and the LSTM only classifies once it physically reaches the surface the strategy requires a network configuration that allows making both types of data compatible to create a common map. The retraining CNN layers get passed to work online through a Frozen feature extractor. Running each model through the map and normalizing the results to make them comparable shows different levels of accuracy against validation data using weakly and fully labeled data on both networks separately as well as the combination in the proposed Self-Supervised structure. This shows results in the range of 85.93 % against validation data for the individual networks and a much more promising 94.61% over weakly labeled data. A better results on the combined approached was excepted and hypothesized since it allows the networks to use different source representing the same class of data.

I. INTRODUCTION

Autonomous ground systems are becoming more ubiquitous in urban and industrial environments but still haven't achieved a more widespread penetration to off-road scenarios. Multiple companies, universities, and institutions already count on a starting point to develop, model, and simulate under urban constraints. Therefore, autonomous capabilities have been developed to conform to a firm basis, particularly for urban commercial vehicles and specific research areas. Leveraging these capabilities and porting them to unstructured, natural, and heterogeneous environments will undoubtedly be a crucial part of the future military strategy. The main problem is that current modeling and simulation tools do not offer a comprehensive and unified approach to autonomy for off-road conditions that allow operationalizing new technologies. This also limits the capacity to drive research and development into military ground operations lead or assisted by autonomous

vehicles. Once a sufficient level of autonomy is achieved, the payoff would be unprecedented since it could partially or entirely remove human operators from being exposed to danger. Mobility is not an issue on the road, but it becomes fundamental to any terrestrial off-road, manned or unmanned system, especially under unstructured, complex or hazardous environments where data-rich sensors provide almost no advantage if a higher-level understanding of off-road conditions is introduced. This is why new techniques have to be developed to extract information from immediate surroundings to assist in navigation tasks. The later needs to go together with accelerating the pace through modeling and simulation tools readily available for on-road vehicles but almost non-existent for off-road, where terrain sensing is one of the most valuable data sources. The focus was not to create an algorithm able to overcome these conditions easily but to measure and determine characteristics from the terrain that would lead to a "No-Go" or restricted condition and capture the data to learn from it applying to a Self-Supervised Deep Learning algorithm able to segment different types of mobility classes and add new ones on the go. The reference vehicle TRACER X is used and modeled from its physical characteristics in order to accelerate the development in the future of artificial intelligence strategies aided by modeling in simulation under off-road conditions. To obtain the simulation, several modern open-source tools are used such as Unreal Engine for graphic simulation, MS AIRSIM as a based connector PhysX as an underlying physics engine. All algorithms are implemented using ROS and the standard NAVStack as an external component connected through AIRSIM. Every node is programmed and treated individually to accelerate development and iterations. The main machine learning hypothesis relies on the combination of Long Short-Term Memory Neural Network to process vibrational data separating it into mobility classes and a Convolutional Neural Networks to classify the same but from a top view, more dense visual data. Both types running together improve the classification results in a Self-Supervised to explore weakly or unlabeled data. The results show that this approach benefits from the data types to increase accuracy in the segmentation and reduce the error in finding the boundaries between terrain types. Figure 1 presents a conceptual overview of the vehicle traversing a simulated terrain and using visual and proprioceptive sensors to classify it in terms if mobility.

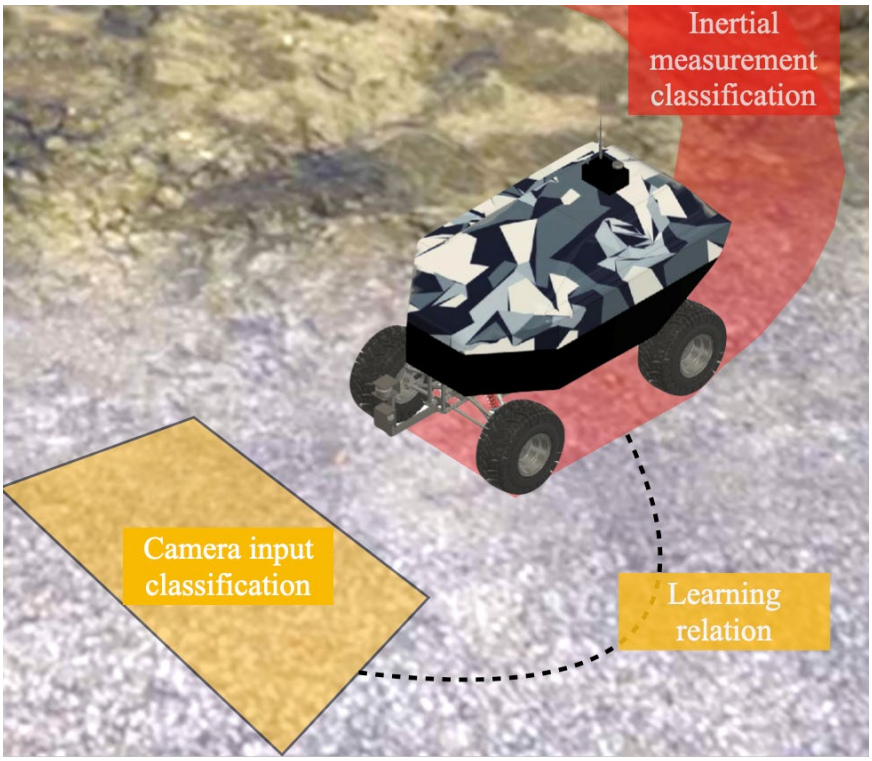


Fig. 1 Digital version of the TRACER X vehicle over a terrain created in Unreal Engine 4 (UE4) and recreating visual and physical properties. This allows the vehicle to explore a map that presents physical and visual properties that relate to the travers ability and possible immobilizations threats. While the vehicle explores, it also creates a map divided by low, medium and high mobility using ORBSlam to locate the surface spatially over a map.

II. TECHNICAL APPROACH

The TRACER X is a small ATV sized autonomous vehicle developed from scratch with the purpose of retrieve data from proprioceptive sensors distributed along with the entire vehicle. These sensors range from inertial, displacement, voltage, current, torque, wheel slip combined with experimental RF sensors and LIDARs to determine front and rear terrain deformation. The tests were conducted under several different conditions such as shallow snow, mud, dirt, slush, and ice and recorded through telemetry and on-board computers based on a NVIDIA Jetson TX2 running ROS. The algorithms were able to detect Go/No-Go conditions using different machine learning approaches where the vehicle came to a stop because of the lack of torque or because of exceeding wheel slip. In both cases, the conditions caused a restriction in mobility. The aim is to develop an algorithm using modern machine learning techniques to determine mobility restrictions before they occur classifying the terrain correspondingly. Several studies have been conducted in the field of mobility exploiting recent developments in Deep Learning applied to planetary exploration and off-road vehicles mostly for small sized and/or slow robots. The design of mechanics, structure, and systems is available and modifiable to serve any specific purpose and in this case fully digitalized with the capability to create verification and validation between simulated and real vehicle. Torque, center of gravity, stiffness, and overall physical properties were measured to make the first approach to simulating the entire vehicle. In terms of the software

interoperability, ROS belongs to the meta-operating system with services like low-level device control, hardware abstraction, implementation of standard functionality, messaging protocols and structures between processes, and package management. ROS follows a service-oriented approach based on nodes that can be invoked. Node communicates over topics, which are messages that are published to the whole system or explicitly directed. Communication is assumed as unreliable and event-triggered, so messages are sent asynchronously and might not always make it to their destination. In terms of sensors, they are implemented as a separate class and collected in libraries. ROS provides a straight-forward hardware abstraction for sensors, so communication, updating, and connecting with other algorithms along the way is relatively simple. The later is a good start for the modeling and simulation stage since it provides a base for further model improvements. Contemporary game engines build upon the components architecture. Game engines are divided into several components, each one providing unique functionality. The combination of open-source robot operating system, physics engines, and gaming platforms is currently being explored by companies, academia, and governments due to the large developing communities, easy access, and cross-platform integration. In comparison to all other modeling and simulation platform Unreal Engine 4 (UE4) stand out for graphics capabilities, including advanced dynamic lighting capabilities and a particle system that can handle up to a million particles in a scene at once by implementing technologies such as *raycast*. Recent changes introduced to the engine refer to highlight, the scripting language for UE4, which in UDK was the UnrealScript language and has now been entirely replaced by C ++ in UE4. In addition, UE4 now uses Blueprints for graphic scripting, an advanced version of Kismet with which you can ultimately assist in the creation of video-games without any previous C++ knowledge. The main structure of the software is composed by several interconnected components with Unreal Engine 4 (UE4) as a base. UE4 provides a complete graphic and physics engine with a native scripting language and graphical interface to create game code and gameplay events with complex interactions through high level objects oriented programming. The connections between Unreal and external elements is provided by the MS AirSim plug-in which is an open-source, cross-platform simulator that supports in-the-loop simulations and the tools to create realistic simulations focused on vehicles and drones. All control, navigation and intelligence algorithms are handled by ROS to maintain the same software structure and compatibility with the physical platform. 3D elements and their interactions are modeled using Autodesk Fusion 360 and then passed to blender to establish interactions, motion range and type, such as steering and suspension. Once the vehicle componentes are modeled, everything is passed to UE4 to establish the physics behind the components as input to the PhysX engine. Interactions, graphics and dynamics are handled by UE4 with the added MS AirSim plugin to integrate a ROS based communication and messaging protocol. Under this scheme, the vehicles sensors and control signals are passed through the ROS messaging to a standard ROS 1 structure handle by nodes.

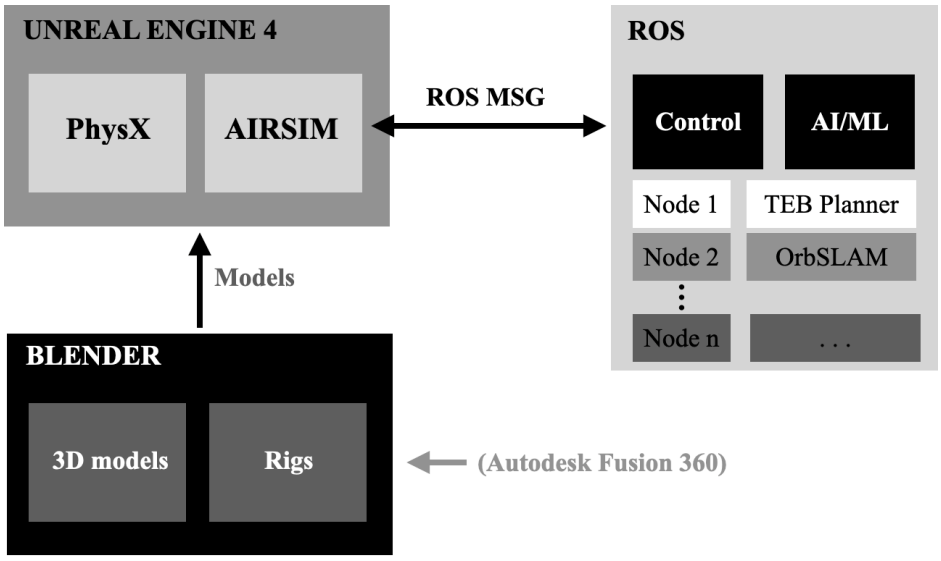


Fig. 2 Software structure for the simulation process considering the elements to create the models, simulate the physics, read sensor and incorporate the control and intelligence strategies

The framework within ROS is structured around the standard Navstack, which contains messaging connectors for planner, sensors, cost map generation, map servers and different sensor sources but lacking the underlaying algorithms to govern them.

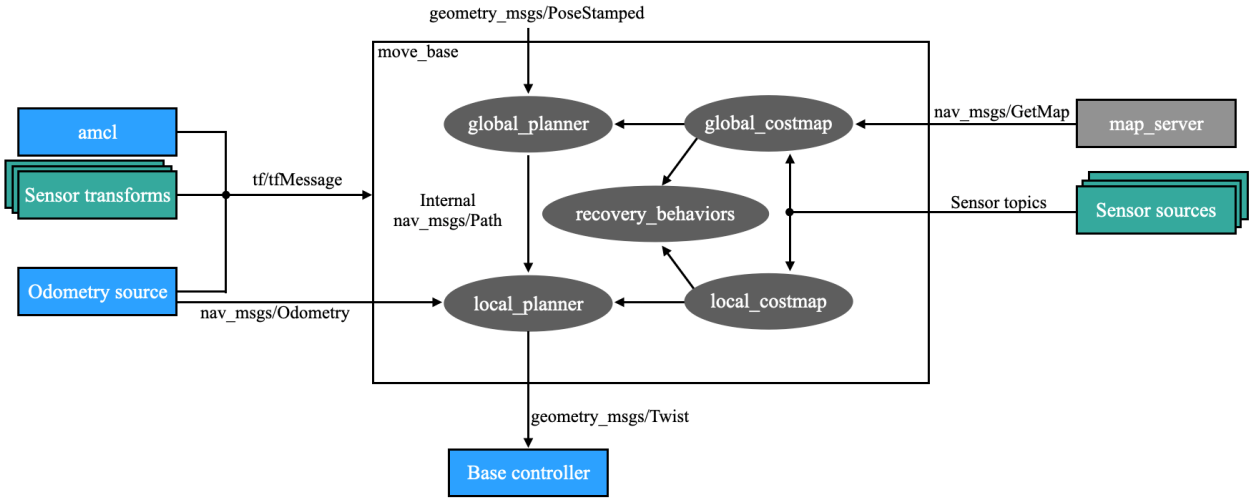


Fig. 3 ROS Navstack base structure

The physical and simulated TRACER X vehicle includes a single front looking camera to navigate. No LIDAR or other exteroceptive sensors assists in this process. It Is therefor relevant to incorporate a strategy to map the surroundings using monocular vision. Intros case, Simultaneous Location And Mapping (SLAM) algorithms provide a proven method to obtain a 3D map using monocular vision and the vehicle movement through the scenario. ORB-Slam specifically allows real-time operation in any size surroundings and incorporates robust ays to counteract motion clutter and occlusion using continuos tracing, mapping, recolonization, and loop closing. The

image is scanned to find and select points on every keyframe which prevail through a survival of the fittest strategy, providing an overview of the 3D scene in front and around. The implementation of this algorithm is under standard parameters. Point cloud data generated from the ORBSlam is shared thought ROS messages to maintain an updated map on local and global map nodes. With this, the vehicle counts with a scenario and mapping capabilities. The navigation problem is solved by the implementation of a Time-Elastic Band. Planner (TEB) with some modifications to work on uneven terrain connecting path planning and control .The planner is used as a collision avoidance method implemented through multi-objective online trajectory optimization. This is highly customizable through parameters to adapt to different needs and flexibility requirements. Navigation connects global targets transforming them into short distance waypoints through the local planner considering the vehicle constraints and obstacles. The surrounding map is recalculated and updated as the vehicles travels through it, deforming the initial global plan with the incorporation of the vehicle's kinematic model and restrictions while updating the local path based on dynamic obstacles and parametrized rules. The Time-Elastic Band creates a sequence of vehicle poses $p_i = (xi, yi, \theta_i)^T$ to modify the global plan. This creates an output which can be used directly as control input commands $\text{cmd}_i = (\text{V}_i, \delta_i)^T$ containing speed (V) and steering angle (δ). In order to allow this planner to work under uneven surfaces, the path estimation is repeated several times modifying the transform from the vehicle reference through ROS, increasing the angle from zero to 45 degrees in 5 degree steps. This means that the algorithm is run in parallel 9 times at different angles to provide a slope traversing estimation. Since all information needs to be integrated at a map level, local and global, OctoMap library is implemented over NAVStack as a 3D occupancy grid providing data structures based on *octree*. This map can take arbitrary environment inputs without prior assumptions representing occupied and unknown areas, free space, and maps with specific information layers all in a probabilistic fashion accounting for noise or dynamic objects. Additionally, this mapping strategy does not restrict on map size and progressive discovery.

A. Virtual environment

To provide the vehicle a scenario to explore, the UE4 tools are used to create a map that incorporates off-road scenarios characteristics such as uneven terrain, vegetation, trees, roads, and most importantly, different type of terrain textures that behave differently in contact with the vehicle. Created maps don't follow particular rules and are not based on a real location but present distributed mobility challenges such as flat hard terrain to low friction inclined surfaces resembling mud. To provide the vehicle with real-time feedback, the terrain is constructed to provide two sources for the sensors to read. The first is a top layer graphical representation of the terrain, which is based on UE4 landscape materials which range from hard concrete type surfaces to highly dynamic snow or sand for each class (high, medium and low mobility) for a total of 6 classes. This is sufficient to give the vehicle a reliable and graphically consistent input for the camera sensor. The second underlying layer is not directly implemented in UE4 and requires a workaround in terms of the landscape mesh that needs to match the terrain graphics. In this sense, a rocky surface needs to considered a mesh to provide the vehicle with an equivalent vibrational signature once it traverses over it. This is introduced to the map on a hidden layer adjusting a randomized floor

sheet restrained by the class. In this sense, the underlying Low Mobility mesh contains a set of superficial image texture combined with a friction coefficient $\mu < 0.3$, a granularity lower than 0.2 [m] and a mesh amplitud of over 3 [m]. The Medium Mobility mesh integrates a friction coefficient of $0.3<\mu<0.7$, a medium mesh granularity of between 0.01 [m] and 0.2 [m] and a medium mesh amplitude of 0.02 [m] to 0.3 [m]. And finally a High Mobility with a friction coefficient of $\mu>0.7$, a high mesh granularity lower than 0.01 [m] and a low mesh amplitude below 0.02 [m]. Figure 4 represents the terrains in graphical layer and parameters. Figure 5 show s a keyframe of the simulation in which the vehicle terrain is being matched for a low mobility, low friction and deformable with underlying properties for vibrational input.

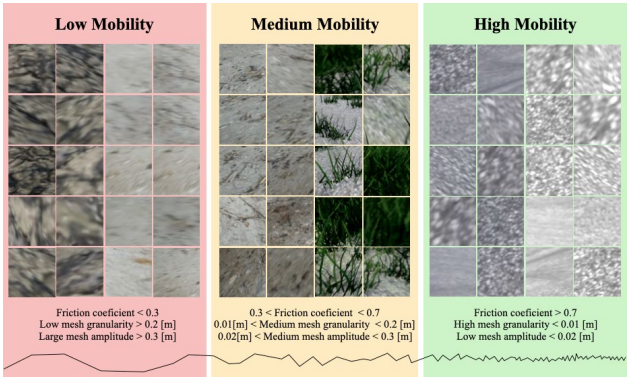


Fig. 4 Three terrain classes divided by graphic and properties layer.

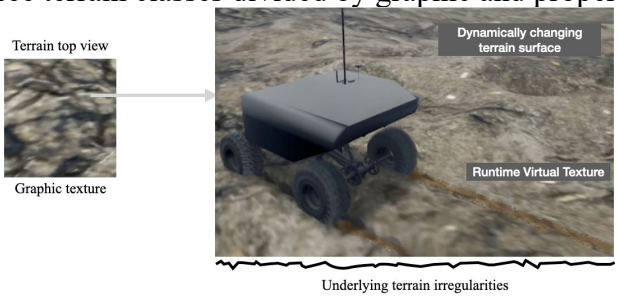


Fig. 5 Simulated vehicle traversing over one type of terrain representing low mobility determined by the graphical layer as something similar to mud, and by the physical layer with deformable properties.

Additionally, and since soft terrain is affected by the contact pressure with the vehicle's tires and could therefor also affect the graphical representation of the terrain, a Runtime Virtual Texture (RVT) is implemented which creates a *texel* data on demand working similar to traditional texture mapping. The RVT is programmed to calculate the area used by the tires at any given time and cache a shading over the specific producing a deformation. This also changes the properties of the terrain altering base color, normal, roughness and specular through a mask

B. Vehicle model

The TRACER X is a small vehicle was designed from scratch using an electric ATV as a base and adding control electronics, sensors, processing capabilities among others. and an overall length of 1.30 [m], 1[m] height and 0.77 [m] wide The design was initially focused on individual components and then integrated to the full structure. Once the integration was ready, everything

was translated into 3D CAD format using Autodesk Fusion360 at scale. The full model was then stripped down form elements not necessary for the simulation such internal components (electronics, actuators, sensors, isolation and others) that weren't relevant from a model perspective. This reduced model was exported as STL (Standard Tessellation Language) to then be imported as a mesh into the Blender software. Within Blender, the model was separated again in its main mechanical and moving components to setup the "bone" structure and introduce mechanic relations and dependencies. Elements such as suspension and wheel are a key aspect for the process since it represents the main interaction with the environment and terrain. Figure 6 shows several stages of the digitalization of the vehicle, starting with the real base, the rigging and connecting of movable parts, the initial integration to a simulated environment for testing, and the final integration to a full graphical environment. Suspension, tire, torque, and physical characteristics governing the vehicle are introduced through the Vehicle Blueprint. The UE4 Vehicle Blueprint consideres a basic input for wheel properties such as shape, damping rate, mass, steering rate, and damping rate. Suspension considers force offset, max raise and drop, natural frequency and damping ratio. These variables were taking directly from a validation and verification process with the real TRACER X vehicle.

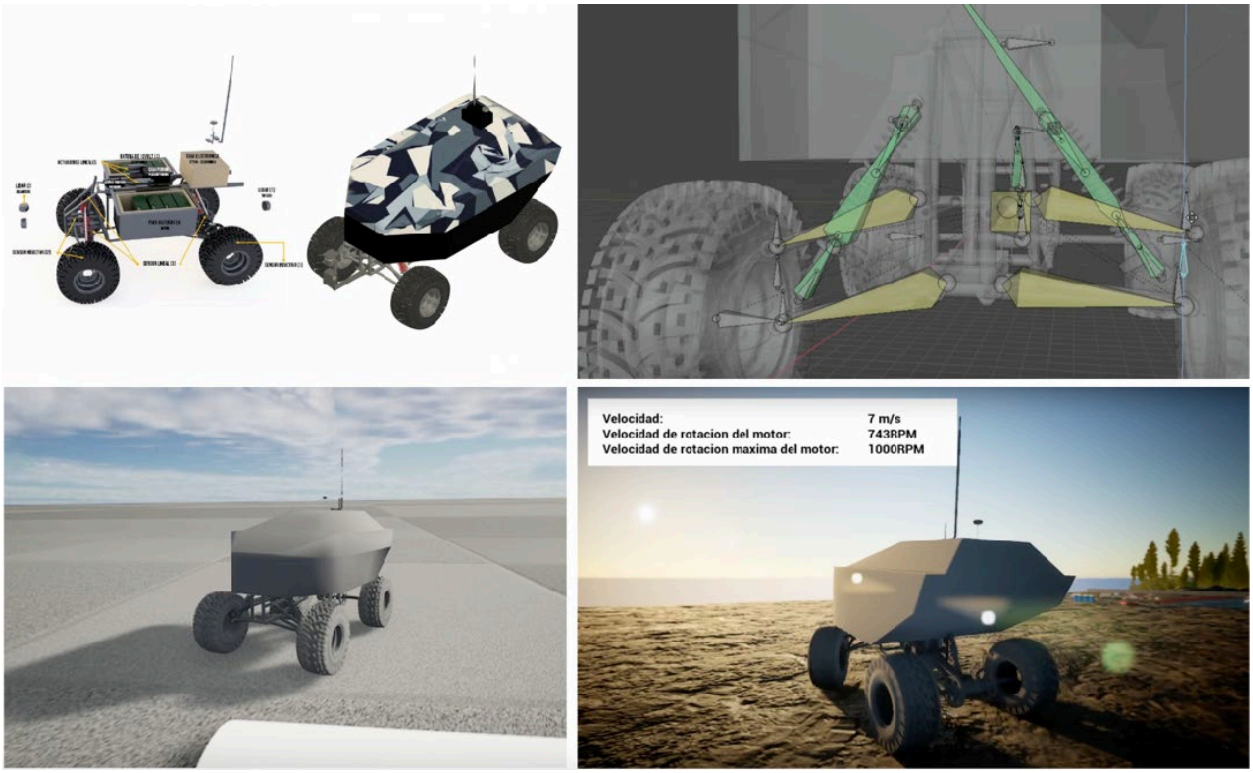


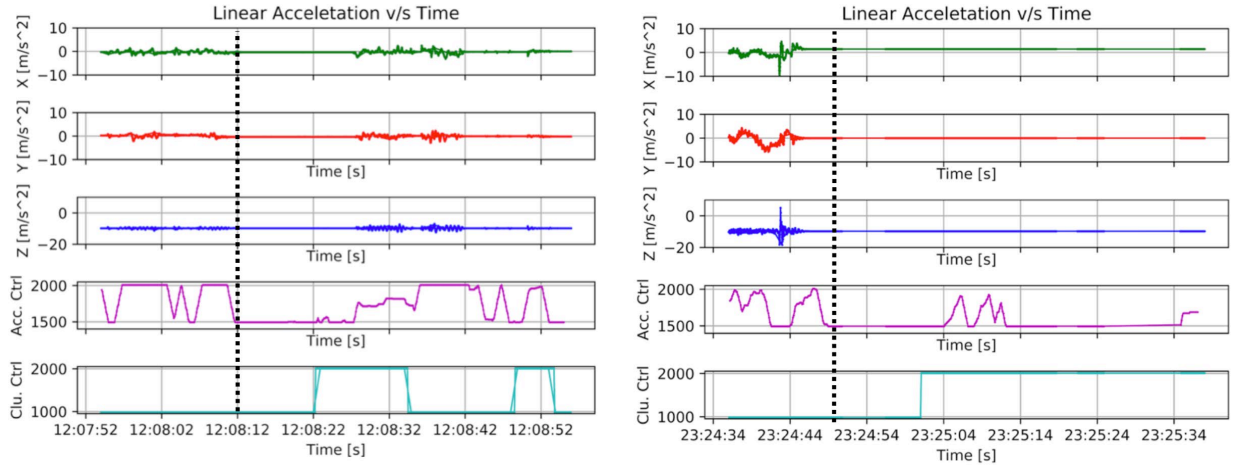
Fig. 6 Top Left: TRACER X 3D CAD based on measures from the real vehicle. Top Right: Blender to determine joints, rotation and mechanical restrictions based on the 3D model. Botton Left: Basic integration of all mechanical features to be tested in virtual environment. Bottom Right: TRACER X full model inserted into Unreal Engine using all PhysX features and graphic capabilities.

Sensors for the vehicle were simulated using the base AirSim characteristics and the integrated advanced version of the PhysX physics engine. These sensors are the internal inertial measurement unit, suspension, RPM, GPS and front looking camera on standard parameters.

C. Proprioceptive terrain classification

In order to obtain a broad variety of trials that differ in lighting, route, and strategy, random point are selected over the map to give the vehicle start and finish points. During the navigation, the vehicle encounters several No-Go conditions. These conditions meant progressive or catastrophic mobility restriction due to wheel slip against lower friction surfaces or insufficient motor torque on large slopes, which produced a hard stop and failed continuation attempts. The data is segmented afterward by Go and No-Go conditions by finding the segments with an accelerator input above zero and a significant reduction in vibrational data on the Z axis below 1/2 RMS or peak motor torque under the same conditions. A deterministic classification algorithm applies these criteria and separates them into Go, traction-related No-Go, and torque-related No-Go conditions. Figure **7** shows a selection of these conditions as detected. The vertical dotted lines represents the NO-Go condition detection where three accelerations are plotted together with the acceleration input (Acc. Crtl) and the forward/reverse input (Clu. Crtl). Once the vehicle hits a the No-Go condition with an unaware navigation algorithm, it will try to free itself to continue. This produces a drop in vibrational input due to the immobilization combined with a sawtooth type acceleration attempts and changes in direction. Figure 8 shows the different vibrational responses to certain type of terrain and the algorithmic determination of what constitutes a Go or NoGo condition by analyzing vibrational data and control inputs, locating points in the data in which the control data doesn't match the output, resulting in mobility restrictions.

Fig. 7 Vehicle on the process of traversing while recoding vibrational data directly from proprioceptive sensors on soft deformable terrain.



(a) (b)

Fig. 8 (a) Decrease in vibrational input due to immobilization combined with failed control acceleration input (Acc. Crtl.). A change in direction (Clu. Crtl) allows the vehicle to momentary regain traction. (b) A sudden drop in vibrational input and failed attempts to accelerate (Acc. Crtl) and change direction (Clu. Crtl) lead to a permanent immobilization.

So the base condition is a reduction in the vibrational input while still providing an input to move through the main motor accelerator or an abnormal torque provided to the traction motor for a defined time window *w*:

Main condition:

$$V_{RMS(t_i)} = \sqrt{\frac{1}{T}\int_i^j v(t)^2 dt} < \frac{1}{2} V_{RMS(w)} = \frac{1}{2}\sqrt{\frac{1}{T}\int_{i+w}^{j+w} v(t)^2 dt} \qquad (1)$$

Once this first condition is met, the Go or No-Go conditions is established by obtaining the motor torque m_{torque} value and compare it to a maximum threshold $m_{maxtorque}$ or and acceleration input from the controller

$$m_{torque} \geq m_{torquethreshold} \qquad (2)$$

Or

$$Accel.Crtl \geq Accel.Crtl_{threshold} \qquad (3)$$

This determines a no-go condition based on the vehicle being incapable of overcoming a slope or large obstacle and when the acceleration rotates the wheel but doesn't get the desired output. A subset of the "go condition" is determined by dividing the threshold by half in both cases:

$$Accel.Crtl \geq Accel.Crtl_{threshold/2} \qquad (4)$$

and

$$m_{torque} \geq m_{torquethreshold/2} \quad (5)$$

Any intermediate segment where the conditions are not met are labeled as “Go condition” and saved for training purposes. A further subset is created to divide the data into a total of three categories. A low mobility in which at least the main and a secondary condition are met, and a subdivision of the “Go” conditions dividing the threshold into half to obtain medium and high mobility. This segmentation resulted in long 5125 datasets and 615 No-Go conditions divided into 205 low, 190 medium, and 220 into high mobility.

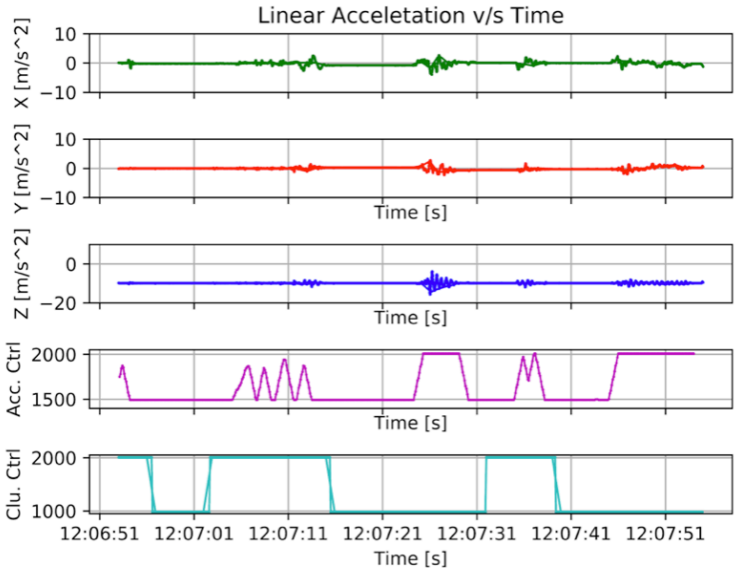


Fig. 9 Normal behaviour without moving forward and reverse without any signs of immobilization to be segmented and labeled as “Go” condition.

This dataset represents the input to train a neural network with information taken programmatically from the simulation. Since recursive neural networks are typically chosen for time serie forecast and classification, a Long Short-Term Memory is implemented for this case.

D. Long Short-Term Memory Recurrent Neural Network

The vehicle needs produce a real-time estimation at a speed that allows decision making before immobilization. Therefor, all vibrational data is incorporated to assist in the classification and labeling of the pre-set conditions of high, medium and low mobility. Available mathematical and data-driven models haven been proven to work in several different types of industrial environments to analyze, classify and label the interaction between the sensor output under different requirements relating spatial and temporal characteristics. Different deep learning methods have entered this space with a broad variety of results. In particular Convolutional Neural Networks (CNN’s) explore spatial relations which has been leveraged for this reason for the vehicle for those properties. But these types of networks lack in temporal relation or dependency between a variable amount of the time steps in relations to previous values composing a long time series, such as the

once present in constant vibrational data. To solve the sequential aspects of the data and relate them a the current state, Recurrent Neural Networks (RNN's) gained traction for time series classification, forecasting and natural language processing, among others, where the input is time-dependent and can change in length. Figure 10 shows a generalized overview of a RNN computational graph.

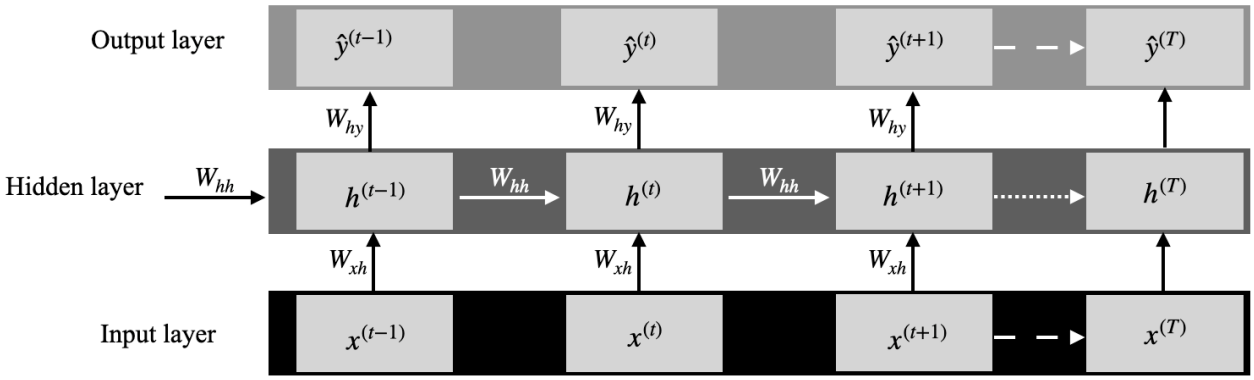


Fig. 10 Generalized structure of a Long Short-Term Memory Recurrent Neural Network

A critical aspect of this configuration relates to the ability to pass previous states through the hidden layers and combining them with the new inputs to create a recursive structure and therefor a time dependency using a recursive structure. This can be described with the following equations:

$$h^{(t)} = f_w(x^t, h^{(t-1)}) \quad (6)$$

$$h^{(t)} = g_1(W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h \quad (7)$$

$$y^{(t)} = g_2(W_{hy}h^{(t)} + b_y) \quad (8)$$

Where W_{xh}, W_{hh}, and W_{hy} are weighted matrices with a temporally shared bias term b_y and b_h, and $h^{(t)}$ represents a hidden state as a function of current $x^{(t)}$ and previous hidden $h^{(t-1)}$ states. This represents the main elements of a RNN type structure. For this structure, back propagation needs to be executed through time (BPT) which generates dependent gradients with respect to h_0 and W_{hh} and creates a long term dependencies.

$$\frac{\delta L^T}{\delta W} = \sum_{t=1}^{T} \frac{\delta L^T}{\delta W_{(t)}} \quad (9)$$

Two problems arise from this. If this gradient becomes too small, new parameters do not add any additional value but forces the network to keep updating irrelevant weights, which is know as "vanishing gradient". And on the other side, the same gradient can become too large with the

resulting weights growing exponentially. A variation to overcome these problems is know as Recurrent Neural Network Long Short-Term Memory, RNN-LSTM to eliminate gradient problems by introducing information flow control by controlling gated recurrent units.

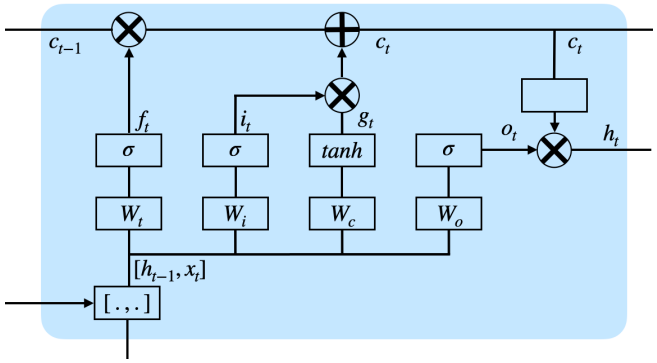


Fig. 11 One LSTM cell with forget, input and output gates.

The main difference is represented by three gates with are forget, input and output. The forget gate allows to control the flow from the previous time step where x_t represent the input at time t, w_{if} the weighted matrix for the input, h_{t-1} is the hidden state at t-1, w_{hf} is the weighted matrix for the hidden state; b_{if} , b_{hf} represent the bias relating input and hidden state. A Sigmoid function is used after the linear combination of weights and biases. This will map the range of f_t between 0, forget information, and 1 to remember. The input gate quantifies the information obtained directly from the input.

$$f_t = \sigma(w_{if}x_t + b_{ij} + w_{hf}h_{t-1} + b_{hf}) \quad (10)$$
$$i_t = \sigma(W_{ii}x_t + b_{ii} + w_{hi}h_{t-1} + b_{hi}) \quad (11)$$

In a similar fashion Wii represents the weight matrix for the input, w_{hi} the weight matrix for the hidden state; b_{ii} and b_{hi} the bias associated with input and hidden state for the input gate. Similarly, a sigmoid function at this gate maps the input between 0 and 1.

$$g_t = tanh(w_{ig}x_t + b_{ig} + w_{hg}h_{t-1} + b_{hg}) \quad (12)$$
$$c_t = f_t c_{t-1} + t_t g_t \quad (13)$$
$$O_t = tanh(w_{io}x_t + b_{io} + w_{ho}h_{t-1} + b_{ho}) \quad (14)$$
$$h_t = o_t * tanh(c_t) \quad (15)$$

Here, w_{ig}, w_{hg} represent the weight matrices for input and hidden state respectively, while b_{ig}, and b_{hg} is the bias associated with input and hidden state for the update of incoming information in cell c_{t-1}. A tanh activation function maps the value of g_t between -1 and 1 which will add or subtract to the new information. The output value uses a sigmoid function to map the value between 0 and 1. The hidden state is obtained using the output and the state of the cell at the specific time with a tanh activation function which establishes the relation between the hidden

state and the long term memory c_t with the output. A Keras LSTM is implemented based on a Tensorflow backend were layers obtain an 3 dimensional array f0 input representing samples, time stamps and features. Samples represents the number of timestamp used to create the time windows necessary to look back or a sequence length and features is the number of features of each sequence per timestamp window.

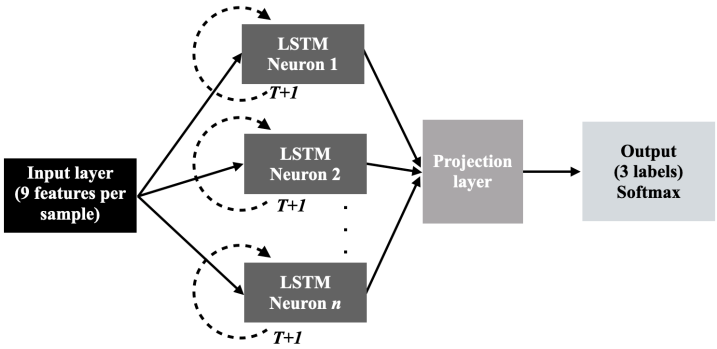


Fig. 12 RNN-LSTM layer configuration

The sequence contains 1 input layer for 9 features representing inertial readings from the vehicle. A second hidden layer with 50 neurons and a 1 neuron output to classify 3 labels. Each input shape will represent a 1 time step with all of the 9 features for each iteration. For this process, Mean Absolute Error (MAE) loss function is applied with an Adam stochastic gradient descent and using 50 training epochs. The model is trained using 50 epochs with a batch size of 72. Internal state are reset at the end of a batch. Since LSTM can easily be overfitted, a Dropout is implemented between layers. This produces a slower convergence during training. The progress in the training converges rapidly as seen in Fig. 13

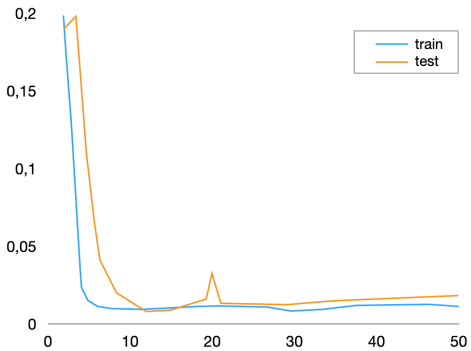


Fig. 13 Line plot of train and test loss during training

Bringing the results into a Confusion Matrix to visualize the results, in which the word represent the ground truth classification and the columns the results from training model, as seen on Table 1

Table. 1 Confusion matrix of the LSTM network on the three terrain classification levels.

	Low	Medium	High
Low	**91.43**	1.4	0.01
Medium	5.3	**87.45**	0.63
High	0.01	0.24	**81.33**

E. Visual terrain classification

Since the front looking camera provides an angle that does not correspond with a top perspective, a geometric projection needs to be provided in order to provide comparable data. This can be done by a simple transformation using the same transformation matrix from the camera sensors. Information from the environment is obtained from the vehicles navigation OrbSLAM algorithms and through warping the input image to provide a birds-eye perspective. For this, the camera calibration *K,* the perspective transformation matrix *P*, the transformation matrix obtained from global coordinates where the frames at t as Tt and the vehicle trajectory Ut can be summarized as

$$u^{(t)} = PKT^{(t)}x \qquad (16)$$

The camera perspective looks ahead pointing down to start the front view at 2 [m] from the most front part of the vehicle on a flat surface. This means that the vehicle will need to travel that distance which will be covered in a time dependent by the speed. On the other hand, the proprioceptive sensors relate to the surface directly under the vehicle. Only once the vehicle reaches a previously classified terrain patch ahead will it be able to match and compare the result of both classifiers. The ground truth will always be provided by the proprioception since it confirms the level of mobility to evaluate the image segmentation. The data to provide weekly labeled base was manually generated using the maps created with Unreal Engine. Sections of the maps were densely annotated for validation purposes. Fig. 14 demonstrates the process in which a generated terrain, with matching physical and graphic characteristics is labeled and segmented into smaller sections obtaining 730 images for validation. The input image size is 500x500x3 RGB matrix from a top perspective.

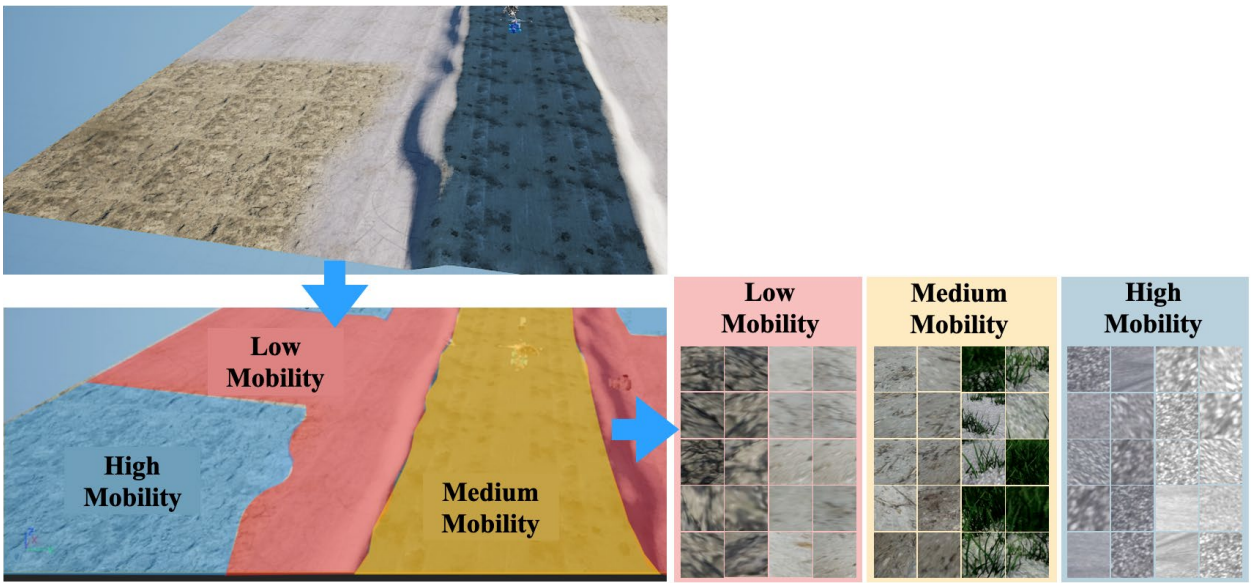


Fig .14 Terrain as created in UE4. Left bottom is the same terrain section but manually labeled without much consideration about borders and precision.Bottom right is the different types of terrain used organized by graphic representation and base mobility parameters.

Convolutional Neural Network (CNN) classifiers are extensively used for all sorts of tasks including segmentation, and classification particularly on spatially related data since they preserve relationships between pixels. In this type of network, every convolution layer organizes hierarchical filter that detect relations and construct among layers. The vehicle used as a reference contains a front locking camera and only the RGB input is being considered and normalized to feed the networks input. A common CNN architecture consists of several convolution layers, pooling layers, activation layers, and fully connected (FC) layers with a final output.

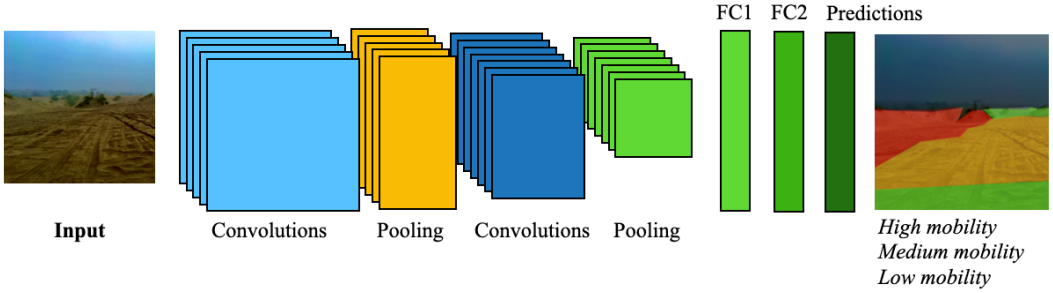


Fig. 15 Common CNN architecture with a gridded input (such as an image), several cascading convolution, pooling and fully connected (FC) layers to obtain a final result.

The structure begins with the x through the first layer which projects to the subsequent layers x_j. W represents a stack of convolution filters in which every layer can be expressed as the sum of convolutions from previous layers, where the * is a discrete convolution operator.

$$x_j = \rho W_j x_{j-1} \quad (17)$$

$$X_j(u,k)j = \rho(\sum_k (x_{j-1})(.,k) * W_{j,k_j}(.,k)(u)) \quad (18)$$

$$(f * g)(x) = \sum_{u=-\infty}^{\infty} f(u)g(x-u) \quad (19)$$

A downsampling step is implemented by max-pooling after the application of the non-linear rectifying function ρ taking the form of a general tensor for W_j. For the implementation, the input is represented by a three channel matrix from the projected RGB image captured by the simulated vehicle. This is then passed to the convolutional layers which scale it down passing it progressively and computing a dot product in between. This configuration is based on the MobileNetV2 network with point and depth wise separable convolutions. The activation layers are based on a non-linear model with a ReLu function max(x,0) in order to implement stochastic gradient descent and back-propagation of error while training. Since the problem at this level represents a multi-class classification, a Softmax function or normalized exponential function is implemented as last layer.

$$\rho(x)_i = \frac{e^{x_j}}{\sum_j^d e^{x_j}}, x = [x_1, \ldots, x_d]^T \quad (20)$$

Back-propagation algorithm for this case allows to use stochastic gradient descent to minimize expected error in the *w* weights and where y_d represents the training target for the *y* output. The separation between both variables *y* and y_d produces the categorical cross-entropy dissimilarity metric *d*, which is the loss function, is usually chosen to be the Euclidean distance, but can be some other measure, such as the categorical cross-entropy, which is the one that we are going to be using in this work.

Error reduction:

$$E(X, w) = \frac{1}{n}\sum_{1}^{\square}\delta(y_\delta - y) \quad (21)$$

Stochastic gradient descent:

$$w_{t+1} = w_t - a_t \frac{\delta E(X, w_t)}{\delta w} \quad (22)$$

Where α_t or the convergence of the stochastic gradient descent algorithms is the choice of the is the stepsizes α_t of the convergence. Finally, to evaluate the performance, classification loss and precision measures are implemented. For semantic segmentation, MobileNetV2 implements a 3x3 depth wise, channel separated convolution, and 1x1 point wise convolutions to extract features. A slight variation of the ReLU function is used to reduce improve performance in which RelU6 ($\rho(x) = \min(\max(0,x),6)$). The last softmax function returns a conditional probability for each data relating it to specific class.

F. Self-Supervised Visual Terrain Classification

The self-supervised algorithms are implemented to learn visual features from the environment using unlabeled data. For self-supervised training, a pretext task solves for the network and pseudo-labels are automatically generated for each image based on some inherited property. The training solves the objective function of the predefined pretext task. Once finished, the learned weights are transferred to a downstream task or used as a feature extractor. The weakly labeled scenarios is use to train the network in a self-supervised fashion. The self-supervisory signal is obtained from images within the robot's close path estimated by the projection and for which the label is known from the unsupervised LSTM classification. This will segment an incomplete mask since the vehicle will unlikely be traveling over the entire map. This consideration is relevant since it means that far background, or any other data points not within the direct path will not increase the segmentation loss. The segmentation loss is formulated for every image by class weighted cross-entropy

$$L_{seg} = \sum_{i}^{C}\sum_{c} w_c y_{i,c} log(p_{i,c}) \qquad (23)$$

In which w_c represents the corresponding weight for each class, $y_{i,c}$ stands for the ground-truth label at a given image coordinate i, and $\log(p_{i,c})$ expresses the *log* probability from the softmax output layer. Finally, the integration of all components are represented in Figure 16

Fig. 16 Full Self-supervised network with the CNN and LSTM working in parallel. The LSTM produces continuos results to provide to the CNN which is passed through a *speed* dependent delay, since the camera is front looking and the LSTM only classifies once it physically reaches the surface for which the robot position and path are required. The retraining CNN layers get passed to work online through the Frozen feature extractor.

VII. EXPERIMENTAL EVALUATION

To create the experimental results, a map was created using procedural placement of trees, vegetation, randomized slopes and terrain patches with textures and physical properties representing the three different classes, each subsided in three, making up for a total of 6 different terrain types. Distance, path, and navigation parameters were also given randomly and left for the TEB planner, ORBSlam, and mapping process to be resolved.This included a start and a target location on the map.

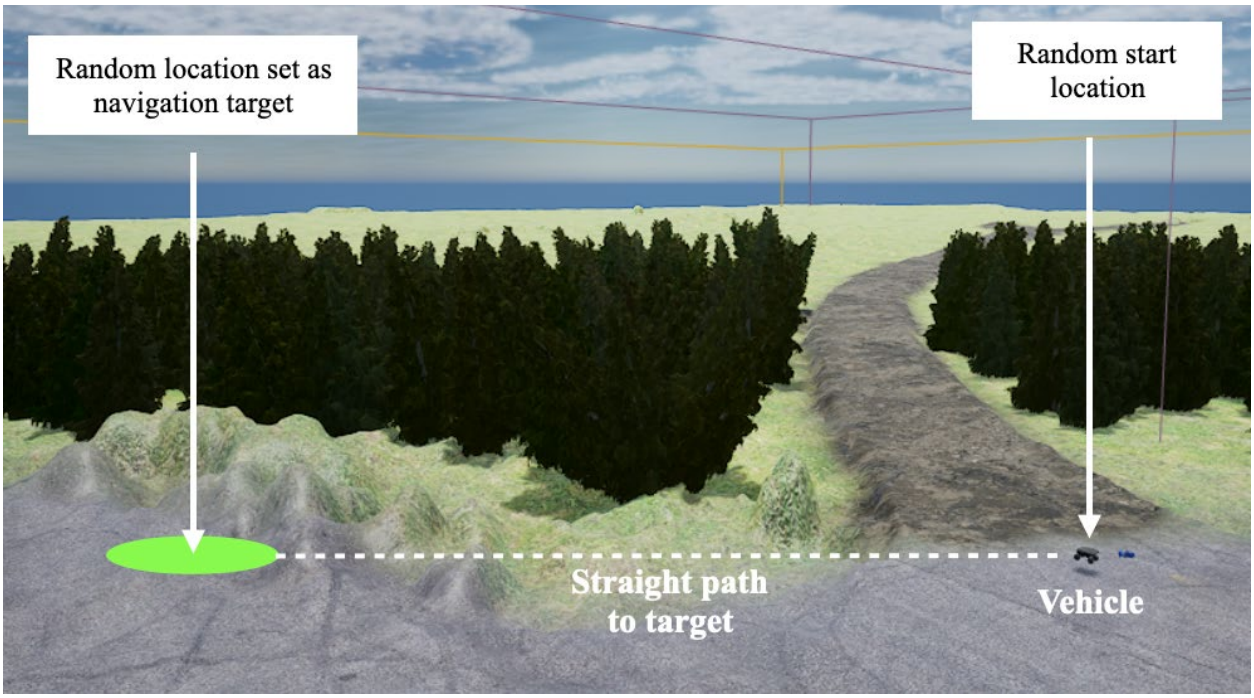


Fig. 17 Random initialization over the map

Fig. 18 Close up rear perspective of the vehicle performing several runs on different types of terrain containing graphic and physics rules.

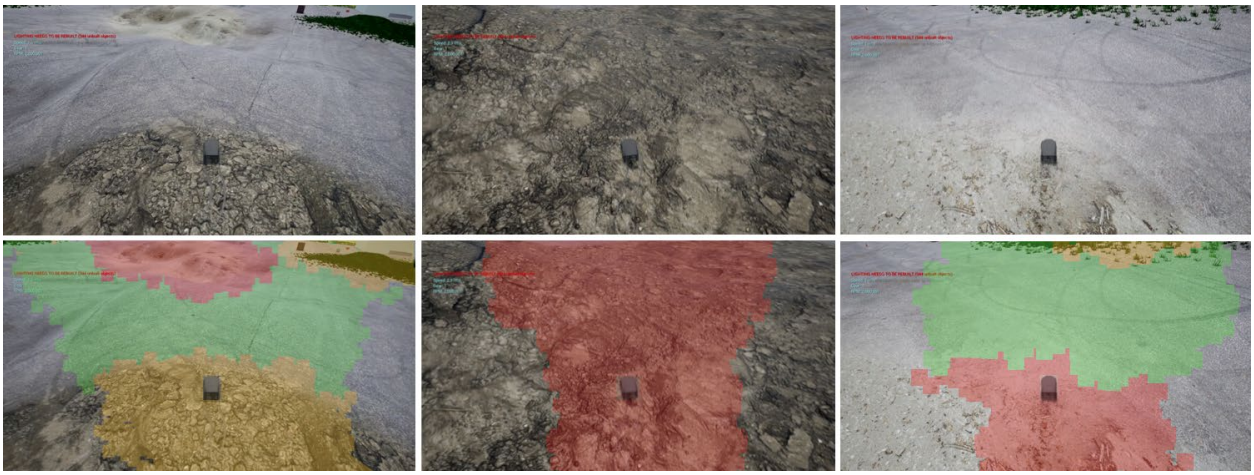


Fig. 19 Simulated vehicle on three different instances in which the top image represents a birds eye view and the bottom the same image with the superimposed mobility estimation map colored red for low, yellow for medium and green for high.

Running each model through the map and normalizing the results to make them comparable the following table shows accuracy against validation data using weakly and fully labeled data and both networks as well as the combination in the proposed Self-Supervised structure. This shows consistent results in both individual networks under supervised training and weakly and fully labeled dataset which averages 85.93 % against validation data. The Self-supervised approach reaches a 94.61% on the weakly labeled data. A better results on the combined approached was excepted and hypothesized since it allows the networks to use different source representing the same class of data.

Table 2. Comparison between individual network evaluation and the combined results under a Self-Supervised scheme .

	Weakly labeled validation (Recall %)			Fully labeled validation (Recall %)		
Mobility	Supervised CNN	Supervised LSTM	Combined Self-Supervised	Supervised CNN	Supervised LSTM	Combined Self-Supervised
Low	78.17	83.43	95.84	91.44	91.43	87.45
Medium	89.72	81.45	96.56	89.23	87.45	89.30
High	80.12	92.33	94.42	85.12	81.33	90.11
MEAN	82.67	85.74	94.61	88.60	86.74	88.95

The results demonstrate that the combined Self-Supervised model represents an improvement over the individually trained networks in any case for the semantic segmentation . One explanation for this is the capability of the combined self-supervised

network to cluster in less noisy levels due to the higher dimensionality of the data and the respective segments. Additionally, an interesting aspect is that the data represented as a map through and created by the vehicle in the exploration is projected into ROS Octomap and can therefor be subtracted from a base ground-truth map. This since the map is created randomly in terms of the different terrain patches but can be mapped using terrain texture or the parameters used to create the terrain and projected into a 2D matrix. Separating mobility classes and not considering areas unexplored by the vehicle this process can be treated as a per class matrix subtraction and normalization, which results in the following table representing the error in the terrain type boundaries.

Table 3. Contrast comparison between map creation and the ground-truth using individual LSTM and CNN networks over fully labeled data and contrasted with the same results on the combined self-learning strategy.

	Ground truth against:		
Mobility	CNN normalized map	LSTM normalized map	Combined Self-Supervised normalized map
Low	0.21	0.15	0.12
Medium	0.12	0.08	0.07
High	0.05	0.03	0.03
MEAN	0.17	0.09	0.07

This results denotes a tendency for the LSTM network to yield better results which could be attributed to a physical contact to the input, unlike the CNN network that needs to determine the boundaries between classes based on graphical gradients. The combined approach improves and reduces the error particularly on the highly deformable terrain class.

VIII. CONCLUSION

The approach leads to several findings in terms of the capability of combined networks to consistently increase they accuracy under a self-supervised learning. The

This work successfully developed robust proprioception-based terrain classification method implementing RNN and CNN network structure under a self-supervised learning model and a completely simulated environment outperforming individual network approaches consistently and over 94% accuracy. This was demonstrated with several runs on the virtual environment recreating different types of scenarios, terrains and its properties in terms of the graphical and physical aspect. This also achieved an important milestone in proving that the fusion of several types of proprioceptive and exteroceptive sensors yield a better results. For this, proprioception RNN and visual CNN were trained independently before for testing and evaluation and contrasted with the combined results. For this case only restricted the amount of terrains where are used for segmentation, but the self supervised method could increase this dramatically and provide a continuous feed off gnu classes that enrich the navigation quality. Additionally the navigation at this point doesn't really make full use of the provided mobility information, so this could be a big improvement for this and a next work.

REFERENCES

[1] "Improving robot navigation through self-supervised online learning," B. Sofman, E. Lin, J. A. Bagnell, J. Cole, N. Vandapel, and A. Stentz, Journal of Field Robotics, vol. 23, no. 11-12, pp. 1059–1075, 2006.

[2] "Deep belief net learning in a long-range vision system for autonomous off-road driving," R. Hadsell, A. Erkan, P. Sermanet, M. Scoffier, U. Muller, and Y. LeCun, in IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008, pp. 628–633.

[3] "Mapping, navigation, and learning for off-road traversal," K. Konolige, M. Agrawal, M. R. Blas, R. C. Bolles, B. Gerkey, J. Sola, and A. Sundaresan, Journal of Field Robotics, vol. 26, no. 1, pp. 88–113, 2009.

[4] C"Self-supervisedterrainclassificationfor planetary surface exploration rovers,".A.BrooksandK.Iagnemma, Journal of Field Robotics, vol. 29, no. 3, pp. 445–468, 2012.

[5] "Terrain characterization and classification with a mobile robot," , L. Ojeda, J. Borenstein, G. Witus, and R. Karlsen, Journal of Field Robotics, vol. 23, no. 2, pp. 103–122, 2006.

[6] "Deep feature learning for acoustics-based terrain classification," A. Valada, L. Spinello, and W. Burgard, in Robotics Research, 2018, pp. 21–37.

[7] "Using sound to classify vehicle-terrain interactions in outdoor environments," J. Libby and A. J. Stentz, in IEEE International Conference on Robotics and Automation, 2012, pp. 3559–3566.

[8] “Deep spatiotemporal models for robust proprioceptive terrain classification,” A. Valada and W. Burgard, The International Journal of Robotics Research, vol. 36, no. 13-14, pp. 1521–1539, 2017.

[9] “The best of both modes: Separately leveraging rgb and depth for unseen object instance segmentation,” C. Xie, Y. Xiang, A. Mousavian, and D. Fox, arXiv preprint arXiv:1907.13236, 2019.

[10] “Multimodal interaction- aware motion prediction for autonomous street crossing,” N. Radwan, A. Valada, and W. Burgard, arXiv preprint arXiv:1808.06887, 2018.

[11] “Modular sensor fusion for semantic segmentation,” H. Blum, A. Gawel, R. Siegwart, and C. Cadena, in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018.

[12] “Towards robust semantic segmentation using deep fusion,” A. Valada, G. Oliveira, T. Brox, and W. Burgard, in Robotics: Science and Systems (RSS 2016) Workshop, Are the Sceptics Right? Limits and Potentials of Deep Learning in Robotics, 2016.

[13] M. Happold, M. Ollis, and N. Johnson, “Enhancing supervised terrain classification with predictive unsupervised learning,” in Robotics: Science and Systems, 2006.

[14] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Muller, and Y. LeCun, “Learning long-range vision for autonomous off-road driving,” Journal of Field Robotics, vol. 26, no. 2, 2009.

[15] K. Otsu, M. Ono, T. J. Fuchs, I. Baldwin, and T. Kubota, “Autonomous terrain classification with co-and self-training approach,” IEEE Robotics and Automation Letters, vol. 1, no. 2, pp. 814–819, 2016.

[16] M. A. Bekhti, Y. Kobayashi, and K. Matsumura, “Terrain traversability analysis using multi-sensor data correlation by a mobile robot,” in IEEE/SICE International Symposium on System Integration, 2014.

[17] S. Zhou, J. Xi, M. W. McDaniel, T. Nishihata, P. Salesses, and K.Iagnemma,“Self-supervisedlearningtovisuallydetectterrainsurfaces for autonomous robots operating in forested terrain,” Journal of Field Robotics, vol. 29, no. 2, pp. 277–297, 2012.

[18] D. Stavens and S. Thrun, “A self-supervised terrain roughness estimator for off-road autonomous driving,” arXiv preprint arXiv:1206.6872, 2012.

[19] L. Wellhausen, A. Dosovitskiy, R. Ranftl, K. T. Walas, C. C. Lerma, and M. Hutter, “Where should i walk? predicting terrain properties from images via self-supervised learning,” IEEE Robotics and Automation Letters, 2019.

[20] M. Nava, J. Guzzi, R. O. Chavez-Garcia, L. M. Gambardella, and A. Giusti, "Learning long-range perception using self-supervision from short-range sensors and odometry," IEEE Robotics and Automation Letters, vol. 4, no. 2, pp. 1279–1286, 2019.

[21] S. Chopra, R. Hadsell, Y. LeCun et al., "Learning a similarity metric discriminatively, with application to face verification," in CVPR, 2005.

[22] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015.

[23] K. Sohn, "Improved deep metric learning with multi-class n-pair loss objective," in Advances in Neural Information Processing Systems, 2016.

[24] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in International conference on machine learning, 2016, pp. 478–487.

[25] X.Guo,L.Gao,X.Liu,andJ.Yin,"Improveddeepembeddedclustering with local structure preservation." in IJCAI, 2017, pp. 1753–1759.

[26] D. Barnes, W. Maddern, and I. Posner, "Find your own way: Weakly- supervised segmentation of path proposals for urban autonomy," in IEEE International Conference on Robotics and Automation, 2017.

[27] N.Hirose,A.Sadeghian,M.Va´zquez,P.Goebel,andS.Savarese,"Gonet: A semi-supervised deep learning approach for traversability estimation," in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 3044–3051.

[28] R. Ku¨mmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard, "Autonomous robot navigation in highly populated pedestrian zones," Journal of Field Robotics, vol. 32, no. 4, pp. 565–589, 2015.

[29] J. Engel, C. Resnick, A. Roberts, S. Dieleman, M. Norouzi, D. Eck, and K. Simonyan, "Neural audio synthesis of musical notes with wavenet autoencoders," in Proceedings of the 34th International Conference on Machine Learning-Volume 70, 2017, pp. 1068–1077.

[30] H. W. Kuhn, "The hungarian method for the assignment problem," Naval research logistics quarterly, vol. 2, no. 1-2, pp. 83–97, 1955.

[31] X. Guo, X. Liu, E. Zhu, and J. Yin, "Deep clustering with convolutional autoencoders," in International Conference on Neural Information Processing, 2017, pp. 373–382.

[32] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.

[34] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," arXiv preprint arXiv: 1802.02611, 2018.

[35] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," arXiv preprint arXiv:1606.02147, 2016.

[36] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "Bisenet: Bilateral segmentation network for real-time semantic segmentation," in Proceedings of the European Conference on Computer Vision, 2018. [37] R. P. Poudel, S. Liwicki, and R. Cipolla, "Fast-scnn: fast semantic segmentation network," arXiv preprint arXiv:1902.04502, 2019.

[38] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, "Erfnet: Efficient residual factorized convnet for real-time semantic segmentation," IEEE Transactions on Intelligent Transportation Systems, 2017.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[40] F. Yu, V. Koltun, and T. Funkhouser, "Dilated residual networks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 472–480.

[41] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1251–1258.