ENHANCING MOBILE MANIPULATION WITH SYNCHRONIZED ARM-LOCOMOTION CONTROL

Virtual Conference 19–23 October 2020

J. Ricardo Sánchez-Ibáñez[1], Raul Domínguez[2], Florian Cordes[2], Carlos J. Pérez-del-Pulgar[1]

[1]Universidad de Málaga, C/Dr. Ortiz Ramos, 29071 Málaga, Spain,
E-mail: {ricardosan, carlosperez}@uma.es
[2]DFKI Robotics Innovation Center Bremen, Robert-Hooke-Str. 1, 28359 Bremen, Germany,
E-mail: {raul.dominguez, florian.cordes}@dfki.de

ABSTRACT

Autonomy on rovers with robotic arms is desirable towards speeding up tasks like sample fetching. This premise is the cornerstone of the work presented in this paper. We describe here the software developed to plan and control the motion of a mobile manipulator, part of the ongoing H2020 project ADE. Its utilization is complemented with the simulation environment MARS, which serves to check the viability of those motion plans created specifically for extreme cases, like when the sample is located on top of a quite difficult terrain. Furthermore, we provide details regarding a series of tests carried out in Bremen with the *SherpaTT* rover, with the purpose of validating the implementation of the software and its use on a case where a hypothetical sample is hardly accessible.

1 INTRODUCTION

Rovers equipped with robotic arms play a key role in the scientific return of planetary exploration missions. Their mobility capabilities allow them to reach more places and as result enhance their possibilities to interact with the environment. In this way, tasks such as sweeping the ground at close range for hyper spectral imaging or sampling ground material can be performed in a higher number of times and situations. Furthermore, upcoming missions foresee the transportation back to Earth of extraterrestrial samples [1]. This is the case in the *Mars Sample Return* campaign, where an ESA rover, the *Sample Fetching Rover* (SFR), shall take those samples left in advance by the NASA rover *Perseverance* and return them to a launcher that later puts them in orbit [2]. Nevertheless, the automatic placement of the robotic arm to carry out these tasks is a challenge. Some research works can be found in the literature that propose different approaches to overcome this problem [3][4]. Up to now, the intervention of ground operators is still mandatory in planetary missions to preserve the integrity of the rover hardware. In the *Mars Science Laboratory* (MSL) mission, for instance, operators make use of a simulation environment named *Surface Simulation* (SSim) to check the effect that may be produced by the commands that are meant to be sent to the Curiosity rover [5].

Figure 1: SherpaTT *with the avionics box and mast installed for the H2020 OG10-ADE project (Autonomous DEcision making in very long traverses, https://h2020-ade.eu).*

In this paper we present a workflow approach that aims at carrying out tasks where a rover and its manipulation arm must move in a coordinated way to reach a specific location or to use the mobile platform to increase the effective workspace of the arm. By combining the arm deployment with the location approach the mission timing can be optimized and important operational time can be spent either for investigating more closely with instruments mounted on the arm or for more terrain coverage per Sol. We propose the use of the simulation step to validate the plan produced by the control software -up to the final motions in a physics simulation- based on the rover's environment representation, prior to sending it to the rover. The main idea consists of the following: a ground operator makes use of the same control software that the real platform runs on-board but connected instead to the rover within a simulation environment. Thereafter, the operator commands the rover in the simulation, sending it to the simulated target location to perform the mobile manipulation task. Then, for any approach operation the control software deliberates two paths to be followed. Firstly,

one that guides the rover mobility system to make the robot get closer to the destination. Later, a second one, serves as a reference for the arm to adopt different configurations according to the robot location, in order to coordinate the motion of both. By completing the operation in the simulation first, the operator has a major degree of confidence to command the same operations on the remote rover. In this way, we provide an additional safety measure to avoid any unforeseen risk, while we take advantage of the same control software that is executed on-board.

2 TARGET PLATFORMS

We make use of MARS (*Machina Arte Robotu Simulans*, https://github.com/rock-simulation/mars), a modular robotics simulator, along with the *SherpaTT* rover, depicted in Fig. 1.

2.1 SherpaTT

This four-legged articulated rover is equipped with a 6DoF robotic arm. The four legs can keep all wheels in permanent ground contact, while actively controlling the roll and pitch of the central body, hence the orientation of the manipulator arm. The *SherpaTT* rover already proved its field readiness in two 4-week field deployments in Utah (2016) and Morocco (2018). The control software for the overall combined platform and manipulator movements is the Mobile Manipulation component developed for the H2020 project ADE and will be tested in a field campaign on Fuerteventura by the end of 2020.

2.2 MARS

The simulation of *SherpaTT* has been developed in MARS. It is a robotics simulator developed and used extensively by DFKI, used traditionally to validate modifications in the robotic software and find and debug any failure. Moreover, it is also convenient to continue a development when the robot is not available or to save time avoiding robotic setups. In addition, the simulator can be used to assess the suitability of certain commands, for instance, to evaluate if the robot would be capable of traversing a certain slope.

In the core of MARS lies the physics engine *Open Dynamics Engine* (ODE, https://www.ode.org/), which provides realistic physics simulations of rigid body dynamics including collisions. Around ODE, a large set of libraries has been implemented that allows the use of convenient features. These include the control of the simulation execution (e.g. starting, stopping, going step-by-step), the load of URDF-based robot models and virtual environments (scenes), the visualization of the simulation execution in real-time and the simulation of the functioning of motors and sensors. MARS software design incorporates a plugin mechanism that eases the addition of new functionalities without modifying the core libraries. The *Control Center* is the central class in charge of the management and communication with the physics engine. Around this class, managers for *Entities*, *Motors*, *Joints*, *Sensors*, *Simulation Nodes*, *Controllers* and *Graphics* provide all needed features for development, testing and identifying advantages and disadvantages of both robot controllers and design.

To simulate a robotic mission in a realistic way, only running the robotics simulator is not enough. A Robotics Control Operating System is needed as well to connect the different software components that run on the system as in the real case. For this reason, the *RObot Construction Kit* or ROCK (http://rock-robotics.org) framework is used to develop tasks that allow the use of the simulation environment in coordination with other robotic software components. This is in fact consistent with the components that will control the low-level mechanisms of *SherpaTT* when running on the non-simulated system.

The *SherpaTT* simulation robot model developed at DFKI is completely compatible with MARS. It includes all the sensors and actuators that the robot currently has, including the avionics box and the mast integrated for the OG10-ADE project. In Fig. 2 is shown this model of *SherpaTT*, placed within the simulated environment that includes the virtual model of the area where the tests described later in this work are made. The simulation environment was produced using a point cloud generated by a drone survey of such area. Furthermore, communications with both simulated and physical robot are possible thanks to the same API, which is shown in the schematic Fig. 3 depicts.

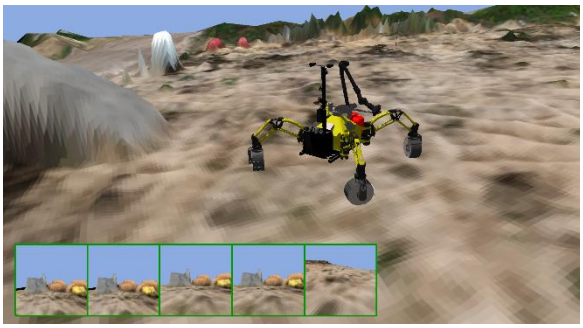


Figure 2: The SherpaTT Simulation includes all sensors available on the rover and the software that controls the simulated rover are the same except for the hardware drivers.

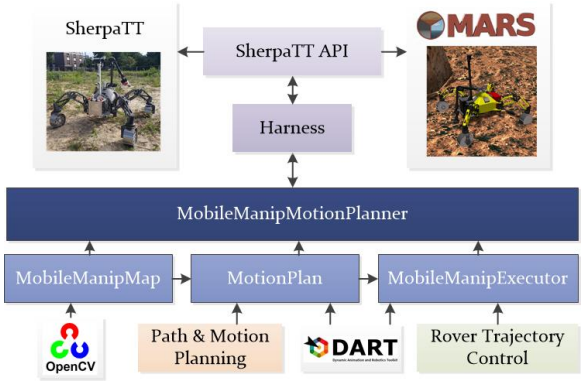


Figure 3: Schematic of the mobile manipulation software architecture, showing its connections with targets and libraries.

3 MOBILE MANIPULATION SOFTWARE

3.1 Architecture

The planning and control software developed for performing mobile manipulation tasks is based on four classes as shown in Fig. 3. The main class, the *MobileManipMotionPlanner*, serves as the interface with the harness component, which in turn uses the API to communicate with either the SherpaTT rover or its virtual equivalent within the MARS environment. This class receives a *Digital Elevation Model* or DEM and creates consequently an instance of the *MobileManipMap* class to handle it. The latter class includes such DEM and oversees the computation of the obstacles and the creation of the cost map that is later used to generate a motion plan. Once a motion plan is deliberated by means of the corresponding class (*MotionPlan*), it can be executed by creating an instance of the *MobileManipExecutor* class, which in turn contains such motion plan.

This software is implemented with the intention of performing two kinds of operation. The first of them is the *Atomic Operation*, which consists of exclusively controlling the arm to make a certain movement, and therefore is out of the scope of this paper. The second operation is the *Coupled Arm-rover Motion Operation*, which must be produced by means of the *Path & Motion Planning* libraries based on the 2D and 3D versions of the *Fast-Marching Method* (FMM) [6]. The latter operation consists of coordinating the rover and the arm in a synchronous fashion to reach the location of a sample and place the end effector on it. Thereafter, it is foreseen the execution of a particular task to do something with the sample, e.g. place the end effector in contact with the sample, pick up a sample of soil or drop it.

The sequence of actions taken to make the rover perform a *Coupled Arm-rover Motion Operation* is explained as follows. First, the harness component calls the *MobileManipMotionPlanner* to instantiate the class. This constructor has the rover surrounding DEM, with all its metadata, as parameter. This DEM is then processed to calculate the obstacles and cost maps. Later on, the harness component requests a motion plan that is based on the initial rover pose (position and orientation) and the estimated sample location. This method generates the rover path and manipulator trajectory using the FMM algorithm. Both depend on the provided information and the *MobileManipMap* object. If the sample can be in fact reached, a *MotionPlan* instance is sent to the *MobileManipExecutor* with the class constructor. Once it is stored, the subsystem would be ready to run the motion plan, which would begin once the harness indicates so.

3.2 Path & Motion Planning

The objective of the motion planner is to reach the sample position and place the rover manipulator close to it, i.e. 10-20 cm far. The proposed algorithm for rover path and arm trajectory generation is, as stated before, based on the FMM algorithm, making use of 2D and 3D workspaces respectively. In a few words, this method computes the numerical solution of a wave that propagates through the environment starting from a certain point. The rate at which the wave propagates depends on the cost assigned to every part of such an environment, which is discretized into a grid. Thus, FMM computes the minimal time at which the wave arrives at each grid node. Then, by making use of the gradient descent method, a path is retrieved from any point to the one from which the wave started expanding. The main advantages of using this method are:

- **Smooth trajectories generation:** unlike other methods like A* or D*, the turning angles of the paths obtained through FMM are not restricted at all. Besides, the location of the waypoints making up these paths are not constrained to the location of the grid nodes, meaning they can be anywhere within the workspace. In this way, it is not necessary to apply any post-processing to the path to smooth it.
- **Optimal solution.** FMM numerically solves the propagation of a wave using the *eikonal* equation, an expression that correlates the propagation rate with a cost value defined at any workspace point. In this way, the retrieved paths always tend to be optimal, and the only error committed is due to the grid discretization. Other grid-search based methods like Field-D*[7] or Theta*[8] cannot ensure this, since, although the computed paths can be also smooth, they make use of

estimation methods that introduce more error and, in some cases, can produce suboptimal solutions.

- **Computer complexity.** It is like other path planning algorithms with fewer features, using a Dijkstra-based grid-search method to visit each grid node. The computer complexity is similar to A* and D*, the most typical path planning algorithms. However, as shown, this method is much better in some features.
- **Parallelization.** Since FMM computes the optimal solution of a wave propagation, by using its bi-directional version it can be parallelized: two waves can be propagated, one originated from the start and the other from the goal location. Then, they encounter at an intermediate point and, because of the nature of FMM, the whole path between start and goal is the concatenation of the path between the intermediate point and the start and between the intermediate point and the goal. This would be useful in the case of using multiple cores processors.

Therefore, the proposed algorithm can generate the rover path and the manipulator end effector trajectory, given a 2D cost map, an initial rover position and the sample location, which corresponds to the desired final manipulator end effector position. Then, by using the inverse kinematics, a profile of the rover joint references can be generated depending on the relation between the rover path and the end effector trajectory. Since the cost map has a direct effect on the resulting path and end effector trajectory, the error committed to build it is here relevant. In this sense, the rover and the sample positions, as well as the DEM, are provided with their respective accuracy. The sum of all estimation errors provided has an impact on the uncertainty of the end effector position as shown in Fig. 5. In this figure, the main reference frames from the rover and manipulator are shown. The first one is the rover position frame with respect to the world frame. Any error on the rover pose is extended to the end effector frame, e.g. a yaw error would increase the end effector position error based on the distance between the rover and the end effector frames (L). On the other hand, an error on the sample location would also increase the total error committed by the manipulator. Taking into consideration these errors, a sphere can be defined. It represents the error space, i.e. the manipulator end effector would be in any place within the sphere. Therefore, the size of this sphere is proportional to the amount of introduced error. Assuming the manipulator has a Force/Torque sensor on the end effector, the vertical error could be reduced by detecting the instant time the manipulator is in contact with the surface. It would belong to the final stage of the manipulator movement.

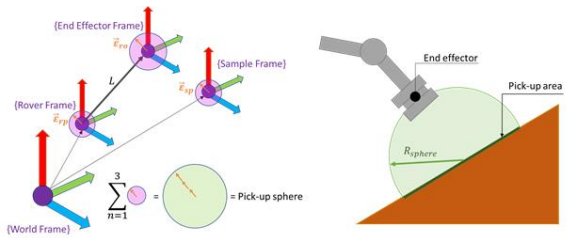


Figure 5: Relation between the estimation errors.

The provided DEM is processed to detect obstacles in the surrounding area of the rover. Two parameters are derived from the elevation data: the slope and the roughness. By means of thresholds values based on previous experiments [9], obstacles can be determined on the map. Thereafter, thanks to the OpenCV library, a metric indicating the distance from any pixel to the closest obstacle can be computed. It serves to produce a cost map that tends to make paths get further from obstacles by means of repulsive fields. The size of these obstacles may be larger due to the amount of estimation error introduced. Therefore, reachability of some samples could be set as unfeasible by the algorithm because of the DEM error, although they could in fact be reached. For example, in Fig. 6 there is a small corridor in position (5,15) that could be closed if there were a big estimation error on the DEM. Therefore, it is important to reduce DEM error to avoid those cases where the algorithm would state that the sample cannot be reached even when it could be in fact.

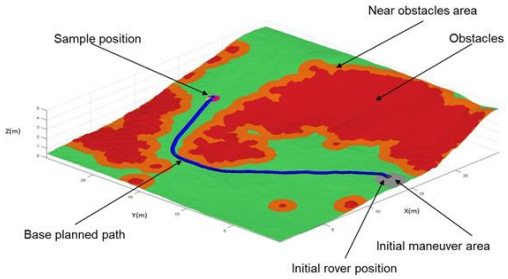


Figure 6: Cost map example.

The FMM does not ensure by itself the arrival of the path following a certain heading final condition. This is because rather than the direction the vehicle is heading, only the 2-D position of the waypoints is considered when computing a path, being the heading of each of these waypoints just the tangent to the path they make up. Moreover, this method does not consider the shape and kinematic configuration of the vehicle, using a simplification in the form of a single point in space. Nevertheless, it is still possible to define a cost map that considers the distance between the rover center and the sample location, while at the same time the rover arrives facing the sample.

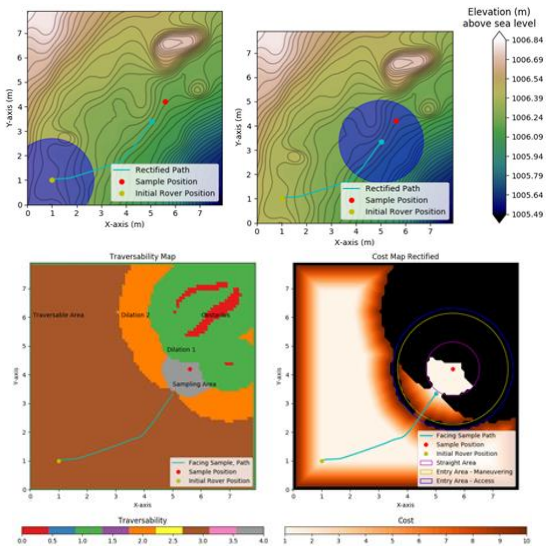


Figure 7: Example case of ensuring the path goes straight to the sample. (Above) The rover, depicted as a blue circle, follows the path towards the sample in the red dot. (Below) Corresponding Traversability and Cost maps created using the DEM.

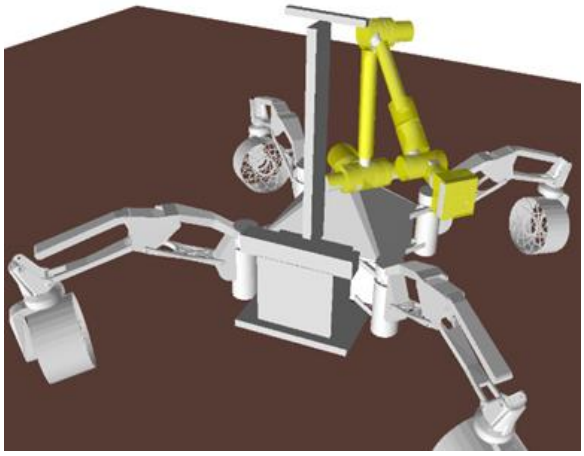


Figure 8: URDF model of SherpaTT (grey) and its manipulator (yellow).

Once the base trajectory is planned, the algorithm generates a new path for the manipulator to reach the sample. During the planning phase, it is necessary to consider possible collisions of the arm with the rover itself (legs, wheels, cameras mast…). To do this, the open source library DART (Dynamic Animation and Robotics Toolkit) is used to detect collisions, together with an URDF model of the whole rover. This model contains the virtual collision objects corresponding to the whole system, as depicted in Fig. 8. By using it, a reachability volume of the manipulator can be generated, which defines what positions of the arm are fully safe. The reachability volume of *SherpaTT* is shown in Fig. 9. If the planner places the arm wrist inside this reachability volume, it is completely ensured that the arm will not collide with the rover. So, to later obtain a 3D path to be tracked by the wrist, a tunnel shaped volume of cost is built surrounding the rover base path, employing the stated reachability volume of the manipulator. An example of this cost tunnel is shown in Fig. 10, where a section of the tunnel shows its interior cost distribution. Basically, the cost is defined in a way it gets higher values while being closer to the limits of the tunnel. In this way, we benefit keeping the wrist as far as possible from the non-reachable areas.

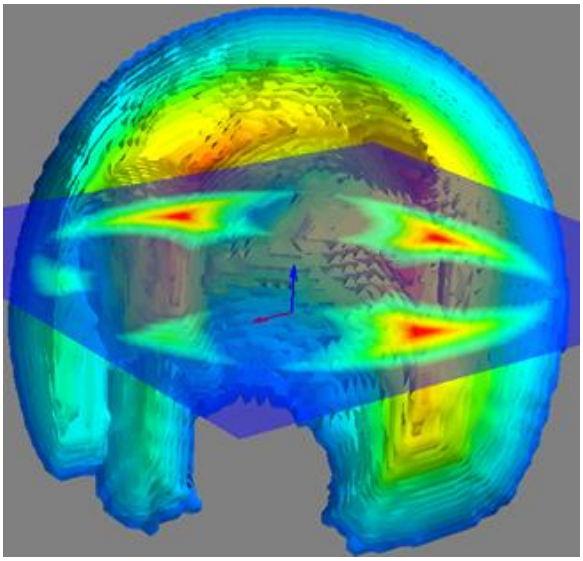


Figure 9: Reachability volume of the manipulator, where reachable zones are colored from red (far from limits) to blue (near limits).

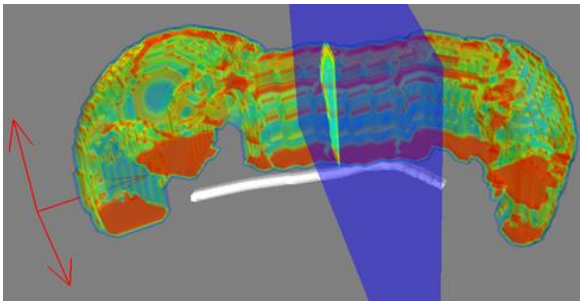


Figure 10: Rover path (white), with the generated tunnel volume associated to it.

Inside this tunnel, the FMM in a 3D version generates a trajectory for the manipulator to reach the sample. An example of a trajectory is shown in Fig. 11, where the initial configuration of the arm is also shown. Next, it is needed to match the manipulator waypoints with the rover planned path. In this stage, it can be configured how the arm is deployed: at the beginning, during the trajectory or close to the sample. Finally,

the arm positions profile is obtained by means of the inverse kinematic model of the manipulator at every waypoint of the trajectory. The end-effector joints positions are set ensuring the last segments of the arm do not collide with anything.

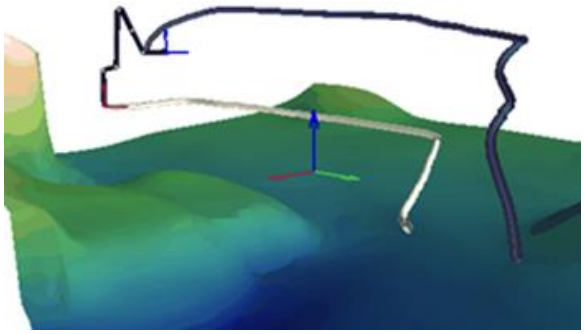
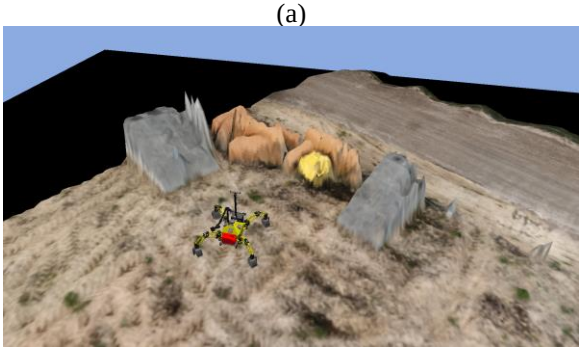


Figure 11: Rover path (white) and arm wrist (blue) trajectories, together with the initial arm configuration (black).

Finally, the *MobileManipExecutor* class serves as the controller that provides the commands to the Sherpa API. According to the state of both the rover and the arm, it returns the rotational and translational velocities of the mobile platform, as well as the position reference of the arm joints. It also makes use of DART to continuously check the status of the rover and prevent any collision during the operation execution. The control algorithm used to control the rover motion is based on the c-pursuit algorithm implemented by ESA [10].

(a)



(b)

Figure 12: Terrain setup chosen for the tests (a) and a virtual model of it built in MARS (b).

4 TESTS

A series of tests were performed to validate the initial version of the component. This implementation is being integrated into the autonomy software of the ADE project. The tests in question were taken in Bremen with *SherpaTT*, in a certain terrain located close to the DFKI facilities, portrayed in Fig. 12a. This terrain is a square area large enough to execute short traverses with a length of few meters. It contains on its corners a series of elements in the form of ramps and tubes that serve as obstacles. The idea behind the first test is to assign the rover the task of going to a certain location next to one of the ramps and place the arm end effector on top of the terrain surface. Thereafter it is emulated an operation to cover the area with the end effector by sweeping it, and later the arm is retrieved and folded.

To preserve the safety of the system, the same tests were also performed in simulation using MARS and the same component software. It was checked the implementation of the mobile manipulation component would behave as expected. Fig. 12b depicts a screenshot of the MARS environment with a virtual model of the terrain. This model was built thanks to the georeferenced images taken from a drone and later processed by the *Pix4D* software (https://www.pix4d.com). Moreover, the input DEM used to feed the *MobileManipMap* class was taken from this processing as well. For solving the localization problem, the position of the rover was obtained by means of an onboard differential GPS antenna.

Fig. 13 presents some pictures showing the *SherpaTT* in action. After a few seconds processing the DEM and deliberating the plan, it proceeds to start moving. Its first action is to unfold the arm, which usually starts being at a predefined *parking* position. Thereafter the mobile platform performs *Ackermann* maneuvers that are combined with the continuous deployment of the arm. The rover reaches a position in which it stops, far enough from the goal location but at the same time close enough to ensure the arm can reach it. In the final step, the rover places the end effector on top of the goal location and starts carrying out a sweeping motion. Such motion serves to cover a certain area, which in turn results from all the uncertainty derived from all the accumulated errors. The main idea behind this is that by executing this sweeping motion we can ensure the arm tip will effectively be in contact with the (fictional in this case) sample. Then, the arm is safely retrieved and left in its initial *parking* configuration.

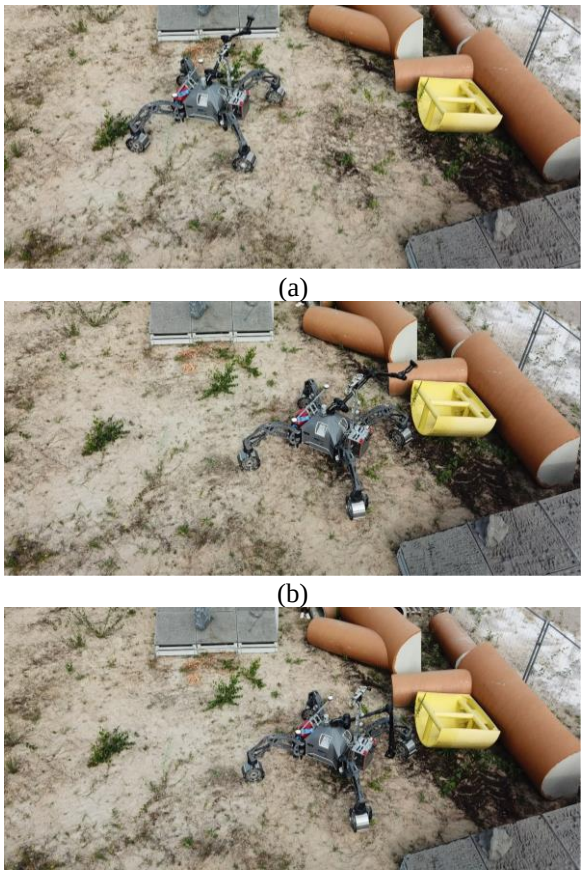


Figure 13: First test done to check the proper functioning of the mobile manipulation software.

The goal of the second test was to prove a hypothetical situation in which it is of great interest to deploy the arm on top of an obstacle. The main idea was to make the rover reach a position that the component would in nominal functioning consider as forbidden, but in fact would be reachable. In the current state of the component, the software would state that there would not exist any feasible plan since it does not allow the rover to get so close to an obstacle. For this particular case, the rover would stop in a place where for certain heading angles its wheels could collide with the obstacle element, so special care is needed for performing this operation. It is worth mentioning, we are not accounting for the reconfiguration capabilities of *SherpaTT* to modify its footprint but considering it as static. Therefore, we lay out this situation to justify the utilization of a workflow including the simulation tool to complement the Mobile Manipulation planning component.

To effectively make the rover reach such a location, obstacles are not considered, i.e. the threshold values for slope and roughness mentioned in the previous section are not considered. This entails the problem of ensuring the rover's safety is not jeopardized. As stated, this is solved thanks to the use of the simulator: it serves to verify that the integrity of the rover would be in fact preserved. Fig. 14 shows how the rover manages to get quite close to the ramp, enough to be inadvisable to turn on the spot, i.e. turn with zero radius. The sweeping operation was slightly modified to make the end effector move at a higher height, on top of the ramp.

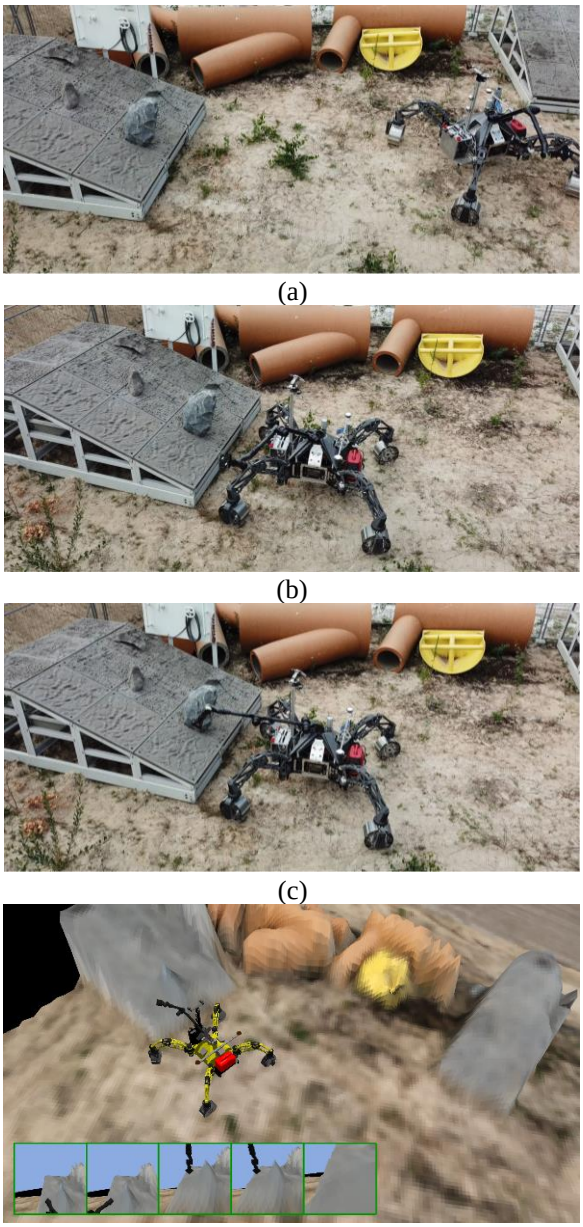


Figure 14: Second test where the plan produced allows the rover to get closer to the ramp. (a-c) It is simulated beforehand to identify the exact parametrization to be used on the operation (d).

5 CONCLUSIONS

This paper serves as an introductory text describing in an overall way the work we are carrying out in mobile manipulation for rovers. We have here explained how the Mobile Manipulation component is built, stressing the workflow created to plan and execute the coordinated motion of both rover and arm. This implementation is integrated within the H2020 ADE project autonomy software. Moreover, a special case in which the MARS simulation is involved to effectively carry out a risky task to make the rover get close to an obstacle is set out. A brief description of the two tests carried out in Bremen are also included. The first served to validate the current implementation of the software, while the second served to emulate a special case in which the simulation verifies the safety of the vehicle in a plan where the thresholds to determine obstacles are artificially removed.

For the near future, it is foreseen that the implementation is further refined and tested. The final goal is to have a mobile manipulation component oriented to sample fetching missions, with the capability to autonomously reach a sample from relatively far in a single run, given the errors that may affect. Some improvements include determining the best position to reach the sample in energetic terms, as in similar research in the past [11], stressing the combined movement of arm and platform by seeing the manipulation as integral part of mobility and increasing the workspace considering the reconfigurable mobile base. Moreover, the arm could be used to enhance mobility by using it actively as a leg.

Acknowledgement

This work is supported by the ADE (Autonomous DEcision making in very long traverses) OG10 project, funded by the European Commission Strategic Research Cluster under Grant Agreement No 821988.

References

[1] Gao, Y., and Chien, S. (2017). Review on space robotics: Toward top-level science through space exploration. *Science Robotics*, *2*(7). https://doi.org/10.1126/scirobotics.aan5074

[2] Muirhead, B. K., & Karp, A. (2019, March). Mars Sample Return Lander Mission Concepts. In *2019 IEEE Aerospace Conference* (pp. 1-9). IEEE. https://doi.org/10.1109/AERO.2019.8742215

[3] Lehner, P., Brunner, S., Dömel, A., Gmeiner, H., Riedel, S., Vodermayer, B., & Wedler, A. (2018, March). Mobile manipulation for planetary exploration. In *2018 IEEE Aerospace Conference* (pp. 1-11). IEEE. https://doi.org/10.1109/AERO.2018.8396726

[4] Liao, J., Huang, F., Chen, Z., & Yao, B. (2019). Optimization-based motion planning of mobile manipulator with high degree of kinematic redundancy. *International Journal of Intelligent Robotics and Applications*, *3*(2), 115-130. https://doi.org/10.1007/s41315-019-00090-7

[5] Verma, V., & Leger, C. (2019, March). SSim: NASA Mars Rover Robotics Flight Software Simulation. In *2019 IEEE Aerospace Conference* (pp. 1-11). IEEE. https://doi.org/10.1109/AERO.2019.8741862

[6] Sethian, J. A. (1999). Fast marching methods. *SIAM review*, *41*(2), 199-235. https://doi.org/10.1137/S0036144598347059

[7] Ferguson, D., & Stentz, A. (2006). Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics*, *23*(2), 79-101. https://doi.org/10.1002/rob.20109

[8] Daniel, K., Nash, A., Koenig, S., & Felner, A. (2010). Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, *39*, 533-579. https://doi.org/10.1613/jair.2994

[9] Cordes, F, Kirchner, F, Babu, A. (2018) Design and field testing of a rover with an actively articulated suspension system in a Mars analog terrain. *Journal of Field Robotics*. 2018; 35: 1149– 1181. https://doi.org/10.1002/rob.21808

[10] Gerdes, L, Azkarate, M, Sánchez-Ibáñez, JR, Joudrier, L, Perez-del-Pulgar, CJ. (2020). Efficient autonomous navigation for planetary rovers with limited resources. *Journal of Field Robotics*. 2020; 37: 1153: 1153– 1170. https://doi.org/10.1002/rob.21981

[11] Pan, S., & Ishigami, G. (2017). Strategy optimization for energy efficient extraterrestrial drilling using combined power map. *IEEE Robotics and Automation Letters*, *2*(4), 1980-1987. https://doi.org/10.1109/LRA.2017.2709912