# Addressing the Entry Barrier for Experimentation in Perception-aware Trajectory Planning for Planetary Rovers

Moritz Kuhne[1], Riccardo Giubilato[1], Daniel Leidner[1] , Máximo A. Roa[1], Senior Member, IEEE

***Abstract*— To fully evaluate perception-aware planning methods for planetary rover, we need a platform that takes action commands and captures perception data. Even with suitable hardware and simulation available to us, there exists an ”entry barrier” for performing research in active vision, as developing methods with a system-in-the-loop is time intensive. We present our approach for tackling this entry barrier by incrementally moving from toy examples to integration and deployment on real robots. Our approach aims at reducing the overall development complexity by producing intermediate results that are used to validate and evaluate the active algorithm.**

## I. Introduction

Perception-aware trajectory planning belongs to the broader scope of active vision. These methods consider perception objectives for choosing the next action and thus the perceived information. Since active vision methods close the perception-action loop, they can't be truly evaluated on prerecorded test data but require an agent to perform the perception-action loop. The authors of [1] argue that simulation and flexible research hardware could be used to fill the gap of a missing system-in-the-loop (SiL). Although hardware and simulators exist for specific platforms [2], [3], [4], there is still the lack of a flexible SiL that captures the diversity of robotic platforms. Furthermore, working with a system is time-consuming. Investigating a failed state or unexpected behavior requires reverting a full system to a former state. Although an agent is necessary to benchmark active vision methods, we reason that development complexity and time can be reduced when carrying out research on active vision methods without a fully-fledged SiL.

We investigate a workflow for concurrently developing and testing a perception-aware trajectory planner for a planetary rover. This workflow is inspired by test-driven software development. New functionality is checked and evaluated against increasingly integrated tests and in environments with growing complexity. Our approach focus on the principles below:

- getting intermediate results and plots
- only implement functionality that one needs
- regression on the existing tests
- evenly distribute the workload over the research time frame

Section II reviews related work. In III, we describe our contribution of a software development process named plot-driven development (PDD). We demonstrate it at the hand of our current work on a perception-aware trajectory planner for the LRU rover, shown in Fig. 1. Section IV discusses the benefits and cost of PDD in the context of developing software for scientific research.

* This work was supported by projects ARCHES (contract number ZT-0033)

[1]The authors are with Institute of Robotics and Mechatronics, German Aerospace Center (DLR), Muenchener Str. 20, 82234 Wessling, Germany `firstname.lastname@dlr.de`

Fig. 1: LRU on Mt.Etna, Italy at the ARCHES demo-mission site where the real data for testing was collected.

## II. Related Work

Our work is inspired by the principles of test-driven development [5]. Whereas test-driven development emphasizes on fully automated testing, we generate plots in an automated manner but delegate the task of inferring correctness of the resulting plots to the researcher. The authors of [6] share our reasoning that checks on visual representations are easily done by a human but hard to package into a test. We, however, deploy this visual test step during development and not in a postponed testing and verification step.

Many approaches test robotic software on a component-level [7], i.e. testing a ROS node. This component-level testing is also shared by tests using a SiL [8], [9]. We, on the other hand, test without a time-consuming SiL and in a more granular approach to gain incremental feedback along the development cycle.

## III. Plot-driven Development

We use PDD as a workflow for developing a perception-aware trajectory planner for a planetary rover. In the spirit of PDD, we tackle the complexity of research in this instance of an active method by developing new functionality against tests.

As the initial action of PDD, we think of visualization that demonstrates the wanted new functionality. We implement the plotting within software tests. These tests produce two results. First, they pass a hard-coded condition. We use conditions to check for assumptions that are easily formulated, e.g. the number of trajectories in collision. Second, the tests create output that is plotted and checked by the researcher. This visualization serves as a test for behavior that is harder to formulate, e.g. the trajectories generated in a receding horizon fashion for reaching the goal form a smooth path. The tests and visual checks pass when the conditions hold and the visualized behavior is accepted by the research. Finally, the tests and plotting are used to regress on when further functionality is added and refactoring is required.

We frame the process of envisioning a plot that demonstrates new functionality and wrapping the execution, assertion, and visualization of the functionality into tests *plot-driven development*. This process is shown in Fig. 2.

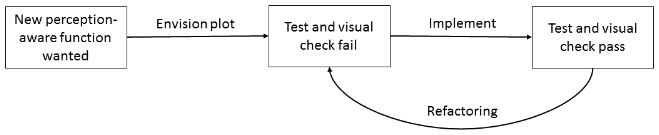

New perception-aware function wanted
Envision plot
Test and visual check fail
Implement
Test and visual check pass
Refactoring

Fig. 2: The process of plot-driven development.

## A. From Toy-Examples to Real-data Test

We use the proposed workflow to develop a perception-aware trajectory planner for the LRU rover. While full integration of the trajectory planner requires a SiL, the development can be based on recorded data. We want this planner to follow a path with motion primitives and track a point of interest (POI) along the global path with forward-facing cameras mounted to the robot base. The final outcome of the development that we describe here should be a envisioned plot, showing a robot that traverses a terrain map gathered at Mt. Etna [10] as shown in Fig 3a. The robot trajectory should follow the global path without large oscillations and closely. This condition is checked visually in the plot. Below we outline the steps of adding this wanted functionality along desired plots.

We use an existing perception-agnostic trajectory planner [11] as a baseline. This planner supports the dynamic model of our robot base, is executed in a receding horizon fashion as required for navigation in unknown environments, and uses the same middleware (ROS [12]) as the LRU. When adding new functionality (here the complete perception-agnostic trajectory planner), we opt for high-level integration test that executes most code at once and adds a meaningful new plot. We start our development by envisioning a plot that shows robot body trajectories which are colored based on their scored of progress towards a goal in a map. We wrap the execution of the existing trajectory planner into a test that runs the planner on a simple world with a single obstacle. We check for progress towards the goal (score) and infeasibility of trajectories with hard-coded conditions. The plot is checked visually. The passing visualization is shown in Fig. 4a.

Next, we require a perception objective, that scores a trajectory on the alignment of a POI with the cameras mounted to the robot body. We envision a plot that visualizes camera frustrums whose colors are determined by their POI-objective. The associated test for tests for the expected costs and outputs the plot shown in Fig 4b.

Following, we want a perception-aware trajectory planner that combines the perception-agnostic trajectory search and the perception awareness. The envisioned plot should show the body trajectories and camera frustrums in colors based on their progress towards a goal and their orientation towards a POI in a map with an obstacle. The implementation integrates the functionalities and visualization elements that were incrementally developed for the succeeding plots. The passing visualization is shown in Fig. 4c.

For the next development step, we want to use the perception-aware trajectory on a model of the LRU with the POI being a cell along the path in a real data map. The envisioned plot is similar to the previous plot (Fig. 4c) but showing the true robot motion and perception capabilities. The accepted plot for this test is shown in Fig. 4d.

Finally, we pass the initially defined visual check of traversing the map of Mt. Etna in Fig. 3a and tracking a POI with the cameras, by looping over the test for Fig. 4d. The result is shown in Fig. 3b.

## B. Comparing Active Methods on prerecorded Data using PDD

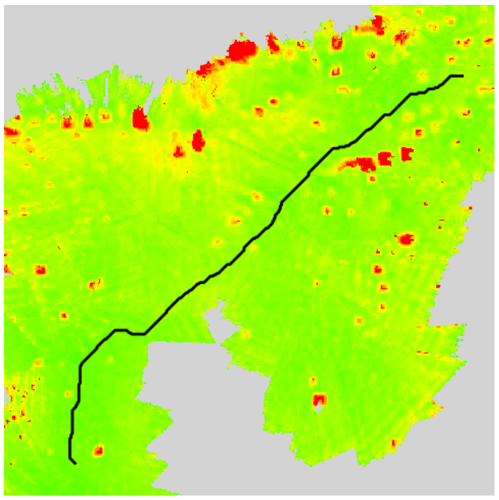
We replay recorded data to compare active methods based on metrics that can be computed at an instantaneous time stamp. For each sampled time step, we evaluate the output of the active methods to be compared. Finally, in line with plot-driven development we wrap this procedure into a test. We demonstrate this at the hand of the perception-agnostic and perception-aware planners from III-A.

We use prerecorded data that was generated by traversing the map in Fig. 3a. At sample times $t_i$, we run the two planners (perception-agnostic and perception-aware) to retrieve their output trajectories. Though not used during planning, we compute the cost $c(t_i)_{POI_{ag}}$ associated to the perception-objective of tracking the POI with the perception-agnostic trajectory. At the same time stamps, we plan a POI-aware trajectory with perception cost $c(t_i)_{POI_{aw}}$. In the test, we check for the constraint $0 \geq c(t_i)_{POI_{ag}} - c(t_i)_{POI_{aw}}$ and plot their difference

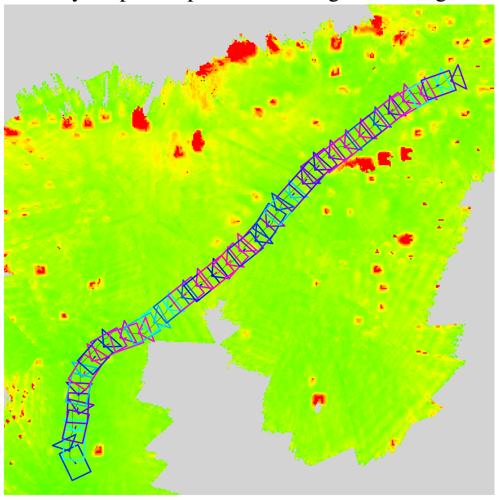
$$\Delta c(t_i)_{POI} = c(t_i)_{POI_{ag}} - c(t_i)_{POI_{aw}}$$

as shown in Fig. 5a.

Special interest lies in the time stamps where $\Delta c_{POI}$ is large. These time stamps indicate the states of robot and environment in which the perception-objective contributes most to the decision making of the perception-aware planner. Fig. 5b shows the POI-aware and POI-agnostic trajectories generated at $t_{max(\Delta c_{POI})}$ for the largest value of $\Delta c(t_i)_{POI}$.

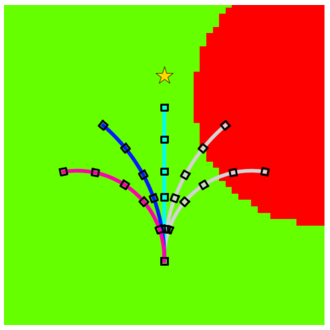


(a) Traversibility map with path connecting start and goal locations.

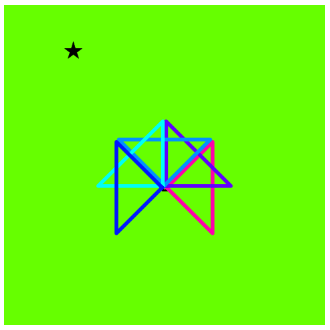


(b) Global path tracking with perception-aware motion primitives.

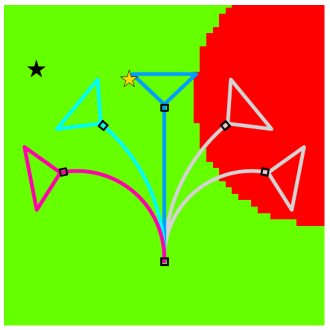
Fig. 3: A 2.5D traversibility map (red are obstacles, green to yellow are small to large costs, grey is unknown space) collected at Mt. Etna with a global path (black line). The trajectories (colored pink to turquoise to distinguish individual trajectories) are generated with a perception-aware trajectory planner using properties of the LRU robot, e.g. size, velocity limits, traversibility.



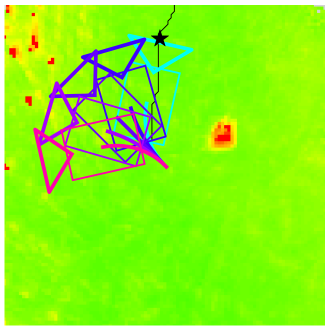
(a) Testing the initial perception-agnostic trajectory planner.



(b) Scoring camera orientation on a perception objective (POI).



(c) Adding perception-awareness to the trajectory planner.



(d) Running the perception-aware planner on a real dataset.

Fig. 4: Integration tests for a perception-aware trajectory planner tracking a global path (black line). Trajectories that collide with an obstacle (red circle) are invalidated (gray trajectory) and the valid trajectories are ranked (turquoise to pink for lowest to highest cost) by progress to the goal (turquoise star) and orientation to the POI (pink star). Trajectories are visualized as the robot's body path (colored line), the robot's body footprint along the path (black square/ CAD model), and the camera frustum (colored triangle) if they are perception-aware.

### C. *Robotic System Integration*

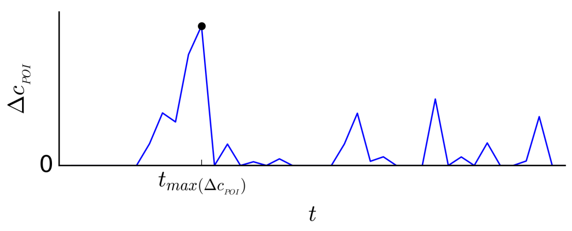
At the time of writing, the perception-aware planner has only been tested without a SiL. We use ROS as a middleware on our target system. The code, developed for Fig. 3 to 5, is encapsulated such that we can test it without a running ROS master or `tf`-look-ups. For the system integration, we add a ROS-wrapper around our planner that makes connections to the ROS master and internally passes on the data and requests.

Since the core functionality is already implemented, we will start by reproducing the highest integrated test on the robot, namely traversing a map with a receding horizon POI-aware trajectory planner. If this first test fails, we plan to investigating the plots in III-A and III-B for the recorded data of the failed test. Since the plots provide initial code locations to search for the erroneous implementations, we expect to integrate the new perception-aware trajectory planner faster when compared to integration without the available SiL-free tests.

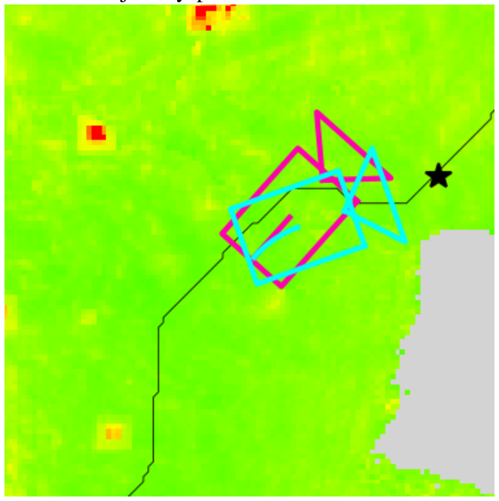
## IV. Discussion

Building test environments, constructing test cases, and writing code for plots are time-consuming tasks. In plot-driven development, we trade in immediate research progress for building this test infrastructure. However, we argue that this is a worthy investment as it allows to continuously reproduce and compare our results. Furthermore, the investment is small, as large portions of the code that are written for the tests can be reused in the next implementations and, most importantly, visualization of the results will also be used to present the research work.

The additional time invested for plot-driven development can be adjusted to the researcher's needs. Testing densely is beneficial in order to quickly discover errors but doesn't



(a) $\Delta c(t_i)_{POI}$ of POI-objective for perception-agnostic and perception-aware trajectory planners.



(b) POI-aware and POI-agnostic trajectories generated at $t_{max(\Delta c_{POI})}$.

Fig. 5: (top) $\Delta c(t_i)_{POI}$ on the replayed data of the test ran in Fig. 3b. (bottom) The trajectory generated at the time $t_{max(\Delta c_{POI})}$ by the POI-aware planner (turquoise) has better POI (black star) tracking than the POI-agnostic trajectory (pink).

produce meaningful plots and comes with large development overhead. As shown in the example in III, adding integration tests, whenever introducing new features, balances well the tasks of supplying enough tests for debugging and also producing meaningful plots. Additionally, by including plots in tests, we found that we will only implement the additional functionality needed to produce the desired plot that demonstrates our scientific contribution.

The biggest weakness of PDD is its poor scaling to large and long-living projects with many researchers. After every change that effects the output plot, an expert needs to evaluate the correctness. This trait also limits PDD in the context of a continuous-integration pipeline. We therefore believe, that PDD is useful to perform research and initial development. However, when moving towards higher technology readiness levels, the plots need to be replaced by a more rigid TDD framework.

## V. Conclusion

Although we have the real robot and a full system simulation available, we still believe that smaller test environments are better suited to progress quickly with our research on a perception-aware trajectory planner. We show that the overhead of building such a test environment can be broken into small steps, that build up to a final test including visualization for presenting their work.

We hope that his workflow helps others to initiate their research in active vision and to gain feedback about their research already along the process.

## Acknowledgment

We thank all the colleagues of the mobile-robotics team, DLR who have helped.

## References

[1] R. Bajcsy, Y. Aloimonos, and J. K. Tsotsos, "Revisiting active perception," *Autonomous Robots*, vol. 42, no. 2, pp. 177–196, Feb. 2018. [Online]. Available: http://link.springer.com/10.1007/s10514-017-9615-3

[2] P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio, and D. Scaramuzza, "Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight," *Science Robotics*, vol. 7, no. 67, p. eabl6259, Jun. 2022. [Online]. Available: https://www.science.org/doi/10.1126/scirobotics.abl6259

[3] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," Jul. 2017, arXiv:1705.05065 [cs]. [Online]. Available: http://arxiv.org/abs/1705.05065

[4] M. Sewtz, H. Lehner, Y. Fanger, J. Eberle, M. Wudenka, M. G. Muller, T. Bodenmuller, and M. J. Schuster, "URSim - A Versatile Robot Simulator for Extra-Terrestrial Exploration," in *2022 IEEE Aerospace Conference (AERO)*. Big Sky, MT, USA: IEEE, Mar. 2022, pp. 1–14. [Online]. Available: https://ieeexplore.ieee.org/document/9843576/

[5] K. Beck, *Test-driven development: by example*, 20th ed., ser. The Addison-Wesley signature series. Boston: Addison-Wesley, 2015.

[6] R. Bocchino, "Industry Best Practices in Robotics Software Engineering."

[7] S. García, D. Strüber, D. Brugali, T. Berger, and P. Pelliccione, "Robotics software engineering: a perspective from the service robotics domain," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Virtual Event USA: ACM, Nov. 2020, pp. 593–604. [Online]. Available: https://dl.acm.org/doi/10.1145/3368089.3409743

[8] F. Weisshardt and F. Koehler, "Automatic Testing Framework for Benchmarking Applications," 2016.

[9] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The Marathon 2: A Navigation System," *arXiv:2003.00368 [cs]*, Jul. 2020, arXiv: 2003.00368. [Online]. Available: http://arxiv.org/abs/2003.00368

[10] M. Vayugundla, M. Kuhne, A. Wedler, and R. Triebel, "Datasets and Benchmarking of a path planning pipeline for planetary rovers."

[11] A. Koubaa, Ed., *Robot Operating System (ROS)*, ser. Studies in Computational Intelligence. Cham: Springer International Publishing, 2016, vol. 625. [Online]. Available: http://link.springer.com/10.1007/978-3-319-26054-9

[12] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System."