Computing the navigation function for the sphere world

Idea

The navigation function φ is the composition of several simpler functions:

$$\varphi = \left(\frac{\gamma_d^k}{\gamma_d^k + \beta}\right)^{\frac{1}{k}} = \frac{\gamma_d}{\left(\gamma_d^k + \beta\right)^{\frac{1}{k}}} \tag{1}$$

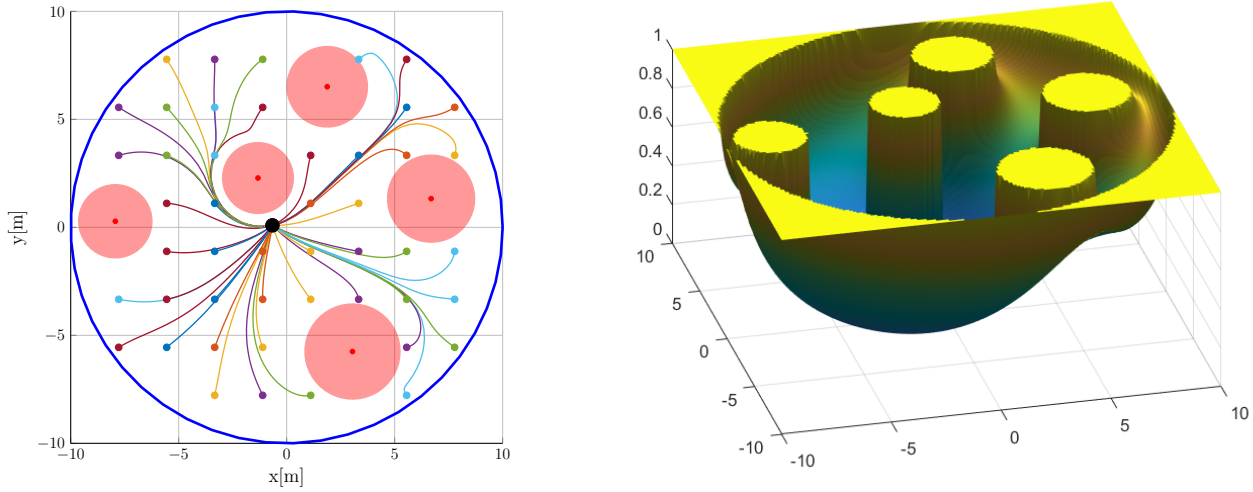
where $\gamma_d : \mathcal{F} \mapsto [0, \infty)$ and $\beta : \mathcal{F} \mapsto [0, \infty)$ are given by

$$\text{attractive component :} \quad \gamma_d = \|q - q_d\|^2, \tag{2a}$$

$$\text{repulsive components :} \quad \beta = \prod_{i=0}^{M} \beta_i, \text{ with } \begin{cases} \beta_0 &= \rho_0^2 - \|q\|^2, \\ \beta_i &= \|q - q_i\|^2 - \rho_i^2, \quad i = 1, \dots, M, \end{cases} \tag{2b}$$

with q_i the obstacles' centers and ρ_i their radii. ρ_0 is the radius of the sphere world and q_d is the target destination.

The navigation function describes a surface that has a single minimum (at the destination) so any agent starting from a feasible point will converge to the destination while simultaneously avoiding the obstacles and the boundary of the world. Trajectories are obtained by computing the gradient at the



(a) trajectories inside the sphere

(b) sphere world with potential surface

Figure 1: Sphere world example

potential surface. For example, single integrator dynamics may be given by

$$\dot{q}(t) = -\nabla\varphi(q(t)). \tag{3}$$

Objectives

There are multiple objectives that may be solved together or in parallel:

O1) Parameter k is given theoretically in [3, 4] but there is no library that actually implements the necessary computations. It should be computed explicitly using CasADi [1] in both Matlab and Python. **[licență/dizertație]**

Theory	Implementation	Tools
●●●●●	●●○○○	Matlab/Python with CasADi

O2) The same ideas may be extended to other families of sets: **[licență/dizertație]**

- star-shaped sets as in [3]
- elliptic/ovoidal sets as in [2]
- polyhedral sets (as collection of linear inequalities)

It would be interesting to implement the same algorithms as done for the classical sphere world case.

Theory	Implementation	Tools
●●●●○	●●●●○	Python and/or ROS2+Gazebo

O3) For a given navigation function implement the control action in either simulation (ROS2 + Gazebo) or experiment (using TurtleBot Burger). **[licență/dizertație]**

Theory	Implementation	Tools
●●○○○	●●●●○	Python and/or ROS2+Gazebo

People

- Florin Stoican
- Theodor Nicu
- Daniel Ioan

Relevant references

[1] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. "CasADi: a software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* 11 (2019), pp. 1–36.

[2] S. Paternain, D. E. Koditschek, and A. Ribeiro. "Navigation functions for convex potentials in a space with convex obstacles". In: *IEEE Transactions on Automatic Control* 63.9 (2017), pp. 2944–2959.

[3] D. E. Koditschek. "The control of natural motion in mechanical systems". In: (1991).

[4] D. E. Koditschek and E. Rimon. "Robot navigation functions on manifolds with boundary". In: *Advances in applied mathematics* 11.4 (1990), pp. 412–442.

Computation alternative for MPI sets for real-time applications

Idea

To ensure the stability of the generic MPC control algorithm (1), we need to consider a terminal set Ω (1d), that is Maximal Positive Invariant (MPI).

$$\min_{\bar{u}_0,\dots,\bar{u}_{N-1}} \sum_{k=0}^{N-1} \bar{x}_k^\top Q \bar{x}_k + \bar{u}_k^\top R \bar{u}_k + \bar{x}_N^\top P \bar{x}_N \tag{1a}$$

$$\text{s.t.} \quad \bar{x}_{k+1} = A\bar{x}_k + B\bar{u}_k, \ \forall k = 0,\dots,N-1, \tag{1b}$$

$$\bar{u}_k \in \mathcal{U}, \quad \bar{x}_{k+1} \in \mathcal{X}, \ \forall k = 0,\dots,N-1, \tag{1c}$$

$$\bar{x}_N \in \Omega. \tag{1d}$$

The main focus of this proposal is to develop a Python toolbox to efficiently compute this terminal MPI set, both for simulations and experimental applications.

For a given LTI dynamics $x_{k+1} = A_\circ x_k$ and a set $\overline{\mathcal{X}} \subset \mathbb{R}^n$ which bounds the state, we have the following standard recurrence for MPI set construction [3]:

$$\Omega_0 = \overline{\mathcal{X}}, \quad \Omega_{k+1} = A_\circ^{-1}\Omega_k \cap \overline{\mathcal{X}}. \tag{2}$$

By construction we have $\Omega_{k+1} \subseteq \Omega_k$, so to determine the MPI it suffices to check either one of:

$$\Omega_k \subseteq \Omega_{k+1} \tag{3}$$

$$\Omega_k \subseteq A_\circ^{-(k+1)}\overline{\mathcal{X}}, \tag{4}$$

$$\overline{\mathcal{X}} \subseteq A_\circ^{-(k+1)}\overline{\mathcal{X}}. \tag{5}$$

To this end, we need to employ several standard notions from set-based control: like polyhedra [1] and/or other particular classes of compact sets. An important aspect is how the representation of these sets may affect the computation time, since all of the stop conditions from (3), (4), (5) will obtain the same MPI set. For example, the most used representation is as bounded and fully-dimensional polyhedra $X \subset \mathbb{R}^d$. The general class of sets has a dual representation, with both a half-space form $X = \{x \in \mathbb{R}^d : a_i^\top x \le b_i, i = 1 \dots n_h\}$ as an intersection of linear inequalities and a convex sum of its extreme points (i.e., its vertices) $X = \{x \in \mathbb{R}^d : x = \sum_{j=1}^{n_v} \alpha_j v_j, \sum_{j=1}^{n_v} \alpha_j = 1, \alpha_j \ge 0$.

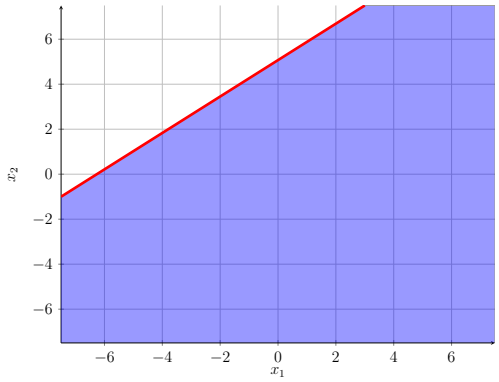


Figure 1: Halfspace (animation, requires Adobe)

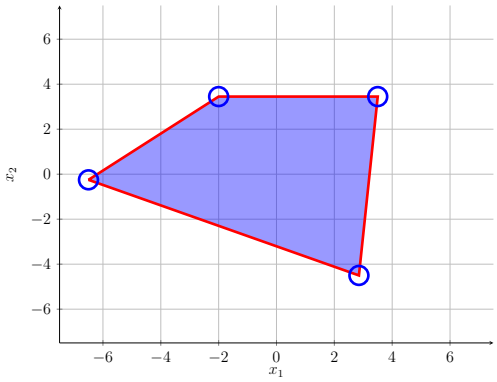


Figure 2: Vertex

Objectives

There are multiple objectives that can be solved sequentially or in parallel:

O1) There is no Python library that actually implements all the necessary computations. It should be computed explicitly using CasADi [2] in Python. **[practică/licență]**

Theory	Implementation	Tools
●●○○○	●●●●○	Python with CasADi

O2) Apply MPC with stability guarantees for the Crazyflie quadcopter in either simulation (ROS2 + Gazebo) and/or experiment. **[licență/dizertație]**

Theory	Implementation	Tools
●●●●○	●●●●●	Python and/or ROS2+Gazebo

O3) Apply MPC with stability guarantees for the TurtleBot Burger in either simulation (ROS2 + Gazebo) and/or experiment. **[licență/dizertație]**

Theory	Implementation	Tools
●●●○○	●●●●●	Python and/or ROS2+Gazebo

People

- Florin Stoican
- Bogdan Gheorghe
- Daniel Ioan

Relevant references

[1] K. Fukuda. "Polyhedral computation". In: (2020). Publisher: Department of Mathematics, Institute of Theoretical Computer Science ETH Zurich.

[2] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. "CasADi: a software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* 11 (2019), pp. 1–36.

[3] F. Blanchini and S. Miani. *Set-theoretic methods in control*. Vol. 78. Springer, 2008.

Graph-based strategies for motion planning

Idea

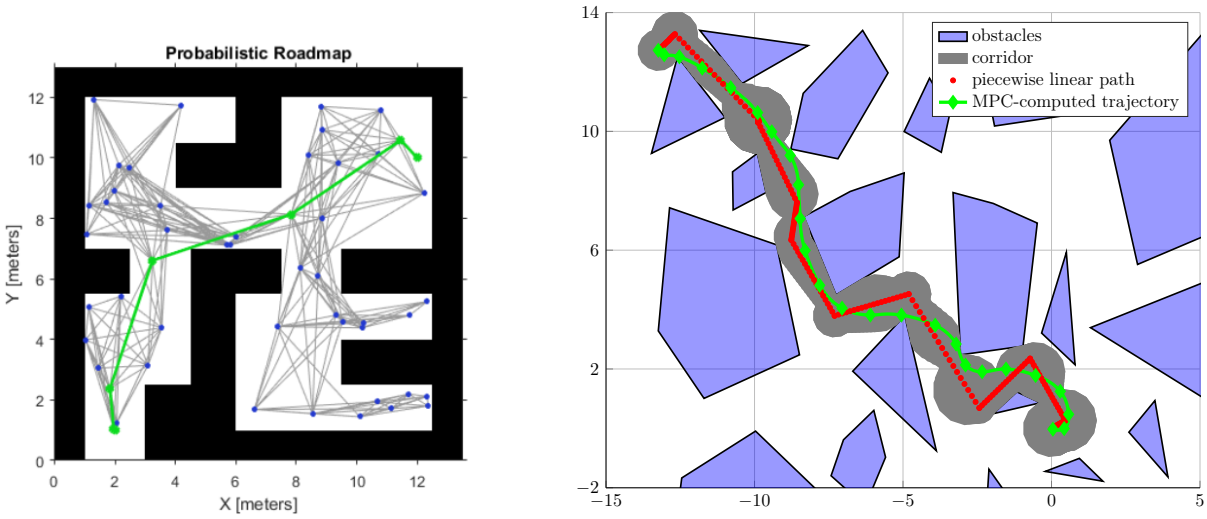
Roughly speaking, in motion planning problems (especially, in obstacle and collision avoidance ones) the challenging part consists in encoding the inherent discrete decisions. The most used approaches in the literature are based on graphs [2]. Thus, those problematic discrete decision are translated to the search of the shortest path between nodes in a graph.

For finding of the shortest path in a graph there exists in the literature various algorithms, but the most influential ones are *Dijkstra's* , *Greedy* or $A^\star$ *search* algorithms.

In the literature, there are two important graph-based algorithms[1]:

1. ***PRM (Probabilistic RoadMaps)*** [4], illustrated in 1a - a multiple-query approach in that it determines the best path through the graph to answer queries after creating the roadmap, which is a rich set of viable paths. Accordingly, if an environment's *awareness map* is readily available, the PRM is a helpful technique [3]. The PRM has many variations, each of which is a significant development.

2. ***RRT (Rapidly-exploring Random Tree)*** [1] - more appropriate in situations where the environment is unknown beforehand. This method builds the graph incrementally, stopping the procedure when a sufficiently big collection of pathways free of collisions is reached. As a result, a collision-free sample is connected to the neighboring nodes and added as a node to the graph. In reality, the resulting graph is a tree. RRT is available in multiple variants as PRM. Some produce just geometric paths that serve as reference trajectories for a lower level controller, while others construct reachable paths while accounting for the equations of motion. Additionally, certain variants are designed for dynamics that are unpredictable, complex, or unstable.



(a) Probabilistic Roadmap method

(b) graph-based method integrated in a MPC strategy

Figure 1: Graph-based approaches for motion planning

The main focus of this proposal is to develop a C++/Python toolbox to efficiently implement this kind of methods, both for simulations and experimental applications.

As well, we plan to extend all this algorithms by incorporating the set theoretic notions right from the stage of designing and constructing the graphs, to ease and to increase the efficiency of the path/trajectory tracking strategies.

[1] The main difference between them is given by the method of constructing the graph.

Objectives

There are multiple objectives that may be solved together or in parallel:

O1) Implement and compare a set of several variants of PRM methods **[practică/licență]**

Theory	Implementation	Tools
●○○○○	●●●●●	C++/Python

O2) Validation of an RRT-type technique through experimentation combined with a conventional path tracking algorithm for the Crazyflie quadcopter or the TurtleBot Burger **[licență]**

Theory	Implementation	Tools
●●○○○	●●●●●	Python and/or ROS2+Gazebo

O3) For an *a priori* known environment implement, test and validate a PRM strategy for the TurtleBot Burger, in either simulation and/or experiment **[practică/licență/dizertație]**

Theory	Implementation	Tools
●●○○○	●●●●●	Python and/or ROS2+Gazebo

O4) For an *a priori* known environment compare in either simulation and/or experiment, a PRM method with a potential field based strategy [5] **[practică/licență/dizertație]**

Theory	Implementation	Tools
●●●○○	●●●●●	Python and/or ROS2+Gazebo

People

- Daniel Ioan
- Bogdan Gheorghe
- Florin Stoican
- Theodor Nicu
- Andreea Udrea

Relevant references

[1] A. Weiss, C. Danielson, K. Berntorp, I. Kolmanovsky, and S. D. Cairano. "Motion planning with invariant set trees". In: *2017 IEEE Conference on Control Technology and Applications (CCTA)*. Aug. 2017, pp. 1625–1630.

[2] J.-C. Latombe. *Robot motion planning*. Vol. 124. Springer Science & Business Media, 2012.

[3] S. Karaman and E. Frazzoli. "Sampling-based algorithms for optimal motion planning". en. In: *The International Journal of Robotics Research* 30.7 (June 2011), pp. 846–894. ISSN: 0278-3649. (Visited on 05/14/2018).

[4] D. Hsu, J.-C. Latombe, and H. Kurniawati. "On the probabilistic foundations of probabilistic roadmap planning". In: *Robotics Research*. Springer, 2007, pp. 83–97.

[5] D. E. Koditschek and E. Rimon. "Robot navigation functions on manifolds with boundary". In: *Advances in applied mathematics* 11.4 (1990), pp. 412–442.

Simultaneous Localization and Mapping (SLAM)

Idea

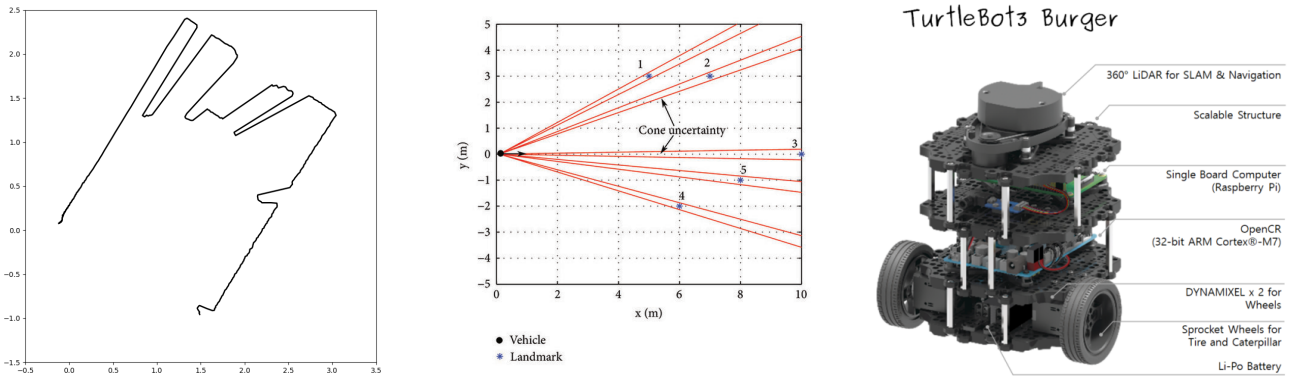
The ability of a mobile robot to navigate through and interact with its surroundings is a critical requirement for a wide range of applications, from autonomous vehicles and drones to rescue robots and household assistants. At the heart of this capability lies the Simultaneous Localization and Mapping (SLAM) problem, which involves a robot concurrently building a map of its unknown environment while also estimating its own location within that map[3].

SLAM has been a long-standing challenge in robotics, with significant progress made over the past few decades. The core idea behind SLAM is to use a robot's sensors, such as cameras, laser scanners, or inertial measurement units, to collect data about the environment and then use this data to construct a map and determine the robot's position within that map. One of the key challenges in SLAM is the need to maintain and update the map over time, especially in dynamic environments where the surroundings may change[3]. Another challenge is the need to disambiguate locations, particularly in environments with similar visual features, such as a warehouse with long, featureless aisles.

To address these challenges, researchers have explored a variety of SLAM approaches, including metric-based methods that aim to build a precise geometric map of the environment, and more recent techniques that incorporate semantic information and unsupervised learning to improve robustness and accuracy[2]. Despite these advances, SLAM remains an active area of research, with ongoing efforts to improve the scalability, robustness, and versatility of SLAM systems to meet the growing demands of modern robotics applications.

The main focus of this proposal is to develop, implement and compare different SLAM strategies that can be exploited in both simulations and experimental applications.

As a starting point we suggest to investigate two standard techniques used in SLAM. The first is based on the use of a(n extended) Kalman filter[5](SLAM problem is formulated as a nonlinear state estimation problem, where the robot's pose and the positions of landmarks or features in the environment are the unknown states to be estimated. EKF provides a recursive solution, allowing the robot to update its belief about its state as new sensor measurements become available), while the second exploit the well-known Interval Analysis methodology[6] (using a Set Inversion Via Interval Analysis (SIVIA) approach and an environment vectorization or combining stochastic and set membership tools). Both techniques make use of data collected by the LIDAR (a remote sensing method -***Li****ght* ***D****etection* ***a****nd* ***R****anging* available on TurtleBot Burger - Figure 1)



(a) Data collected by the lidar (b) Localization uncertainties[4] (c) TurtleBot 3 Burger

Figure 1: SLAM Approaches and experimental platform

Objectives

There are multiple objectives that may be solved together or in parallel:

O1) For a given partially known environment implement an EKF-based SLAM strategy **[practică/licență]**

Theory	Implementation	Tools
●●●○○	●●●●○	Python

O2) For a given partially known environment implement an Interval-based SLAM strategy using CODAC[1] **[practică/licență]**

Theory	Implementation	Tools
●●●○○	●●●●●	Python with CODAC

O3) Experimental validation of an EKF-based SLAM strategy for the TurtleBot Burger, in either simulation and/or experiment **[licență/dizertație]**

Theory	Implementation	Tools
●●○○○	●●●●●	Python and/or ROS2+Gazebo

O4) Experimental validation of an Interval-based SLAM strategy for the TurtleBot Burger, in either simulation and/or experiment **[licență/dizertație]**

Theory	Implementation	Tools
●●○○○	●●●●●	Python and/or ROS2+Gazebo

People

- Daniel Ioan
- Florin Stoican
- Bogdan Gheorghe

Relevant references

[1] S. Rohou, B. Desrochers, and F. Le Bars. "The Codac Library". In: *Acta Cybernetica*. Special Issue of SWIM 2022 (Mar. 2024). DOI: 10.14232/actacyb.302772. URL: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/4388.

[2] O. Çatal, T. Verbelen, T. Van de Maele, B. Dhoedt, and A. Safron. "Robot navigation as hierarchical active inference". In: *Neural Networks* 142 (2021), pp. 192–204.

[3] M. Servières, V. Renaudin, A. Dupuis, and N. Antigny. "Visual and Visual-Inertial SLAM: State of the Art, Classification, and Experimental Benchmarking". In: *Journal of Sensors* 2021.1 (2021), p. 2054828.

[4] Z. Wang and A. Lambert. "A Low-Cost Consistent Vehicle Localization Based on Interval Constraint Propagation". In: *Journal of Advanced Transportation* 2018.1 (2018), p. 2713729.

[5] G. P. Huang, A. I. Mourikis, and S. I. Roumeliotis. "Analysis and improvement of the consistency of extended Kalman filter based SLAM". In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 473–479.

[6] L. Jaulin, M. Kieffer, O. Didrit, et al. *Interval analysis*. Springer, 2001.

Formation control for a team of robots/drones

Idea

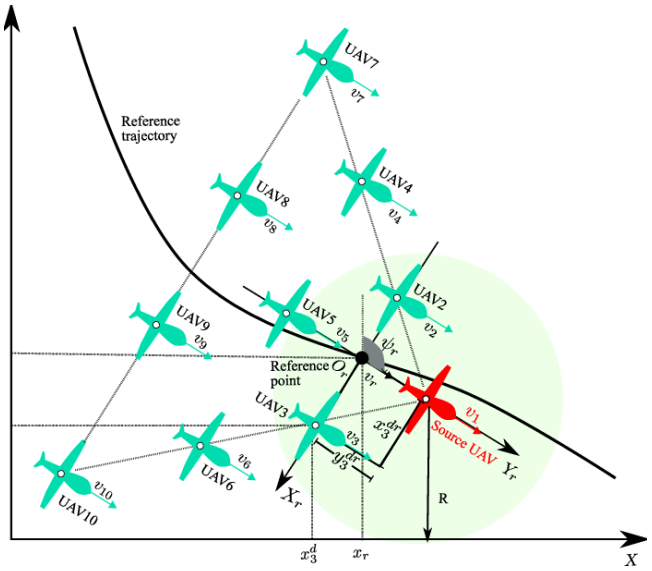
The field of multiagent systems has seen a growing interest in recent years, with researchers exploring various aspects of coordinating and controlling the behavior of multiple autonomous agents. One critical area within this domain is formation control, where the goal is to enable a group of agents to maintain a specific geometric arrangement or shape as they navigate their environment.

Formation control in multiagent systems has applications in diverse domains, such as robotics, aerospace, and transportation. The ability to coordinate the motion of a group of agents while preserving a desired formation can be useful in tasks like search and rescue operations, environmental monitoring, and collaborative manipulation of objects. Researchers have explored various approaches to formation control, drawing inspiration from game theory and reinforcement learning techniques.

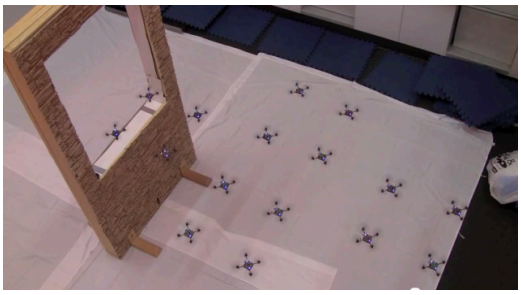
Depending on the sensing capabilities and interactions between agents [2], the formation control algorithms can by classified as:

- Position-based control: agents sense their own positions with respect to a global coordinate system (no agent interaction). This is efficient, but the neglect of interaction might have undesired effects (i.e., collisions and crash).
- Displacement-based control: agents actively control displacements of their neighboring agents to achieve the desired formation. This is advantageous in terms of the sensing capability, but it is more expensive (computational and/or financially)
- Distance-based control: inter-agent distances are actively controlled to achieve the desired formation. Here, we have a trade-off between sensing capability and interaction.

The main focus of this proposal is to create a Python toolbox for controlling a formation of robots or drones.



(a) Formation of drones



(b) Formation of nano-drones Crazyflie 2.0

Figure 1: Graph-based approaches for motion planning

Example formation control

Here we have the single integrator case, where we control the distance between two agents (i.e., drones/robots). The state of the system is given by the position of the agent, and the input is the velocity of the agents.

- Dynamics:

$$\dot{p}_i = u_i, \qquad i = 1, \dots, N, \tag{1}$$

where p_i is the state of the system, and u_i is the input of the system.

- Control law:

$$u_i = k_p \sum_{j \in \mathcal{N}_i} \omega_{ij} \left(p_j - p_i - p_j^\star + p_i^\star \right), \tag{2}$$

where $p_i^\star$ is the reference position of the agent.

Formation requirement: $p_i - p_j \to p_i^\star - p_j^\star$.

Objectives

There are multiple objectives that may be solved together or in parallel:

O1) Implement formation control algorithms in the simulation. **[licență/dizertație]**

Theory	Implementation	Tools
●●●●○	●●●●○	Matlab/Python and/or ROS2+Gazebo

O2) Use the formation control algorithms to follow a given trajectory for each robot or for the entire formation. **[licență/dizertație]**

Theory	Implementation	Tools
●●○○○	●●●●●	Python and/or ROS2+Gazebo

O3) Study the stability properties of different control laws employed in formation control [2] **[licență/dizertație]**

Theory	Implementation	Tools
●●●●●	●●●○○	Matlab/Python with CasADi

O4) Implement the MIP-algorithms in [1] for a group of minimum 3 agents. hfill **[licență/dizertație]**

Theory	Implementation	Tools
●●●●●	●●●●○	Matlab/Python with CasADi

O5) Implement a potential field algorithms for a group of minimum 3 agents. **[licență/dizertație]**

Theory	Implementation	Tools
●●●●●	●●●●○	Matlab/Python with CasADi

People

- Florin Stoican
- Daniel Ioan
- Bogdan Gheorghe

Relevant references

[1] I. Prodan, F. Stoican, S. Olaru, and S.-I. Niculescu. *Mixed-integer representations in control design: Mathematical foundations and applications*. Springer, 2016.

[2] K.-K. Oh, M.-C. Park, and H.-S. Ahn. "A survey of multi-agent formation control". In: *Automatica* 53 (2015), pp. 424–440.

Robot simulator review

Idea

The idea of this proposal is to do a hands-on survey on different robot simulators. We are mainly interested in finding which robot description language is compatible with the most commonly used robot simulators in research/industry. Another interest result would be finding if a sensor fusion model can be imported in multiple robotic simulators.

The robot simulators that we are interested in:

- Isaac Sim: is a robot simulator developed by Nvidia. It's specific advantage comes from the direct access to the GPU resources, make it efficient for intensive simulations. See: Pegasus Simulator based on Isaac Sim.

  Documentation: https://docs.omniverse.nvidia.com/isaacsim/latest/index.html

- Webots: is an open-source robot simulator that has multiple interesting tools, and allows programming in multiple languages.

  Documentation: https://cyberbotics.com/doc/guide/foreword

- OpenAI Gym - free Python toolkit

  Documentation: https://github.com/Farama-Foundation/Gymnasium

- Coppelia Robotics: allows for fast robotic implementation for simulation.

  Documentation: https://www.coppeliarobotics.com/

- Other simulators: jMAVSim, RotorS Gazebo, PX4-SITL, AirSim, Flightmare, MuJoCo, etc...

Objectives

O1) Develop a robot with it's kinematics, and simulate in all the simulators **[practică/licență]**

Theory	Implementation	Tools
●●●●○	●●●●○	Python/C++/XML

O2) Test the sensor fusion capabilities inter platforms **[licență/dizertație]**

Theory	Implementation	Tools
●●○○○	●●●●○	Robot simulators

People

- Florin Stoican
- Bogdan Gheorghe
- Daniel Ioan

The point location problem in the explicit MPC implementation

Idea

Consider the linear time-invariant (LTI) discrete system:

$$x_{k+1} = Ax_k + Bu_k, \quad y_k = Cx_k.$$

Then the typical (quadratic cost and linear constraints) MPC problem is:

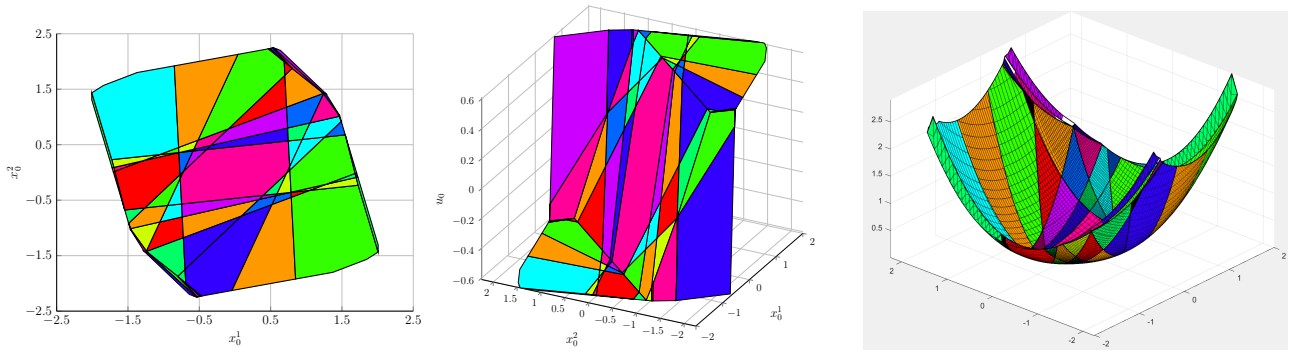
$$\begin{aligned}
\mathbf{u}_N^\star = \arg\min_{u_N} \; & x_N^\top S x_N + \sum_{k=0}^{N-1} \left(x_k^\top Q x_k + u_k^\top R u_k \right), && \leftarrow \text{quadratic cost} \\
\text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k, && \leftarrow \text{state equation} \\
& y_k = Cx_k, && \leftarrow \text{output equation} \\
& x_k \in \mathcal{X}, u_k \in \mathcal{U}, y_k \in \mathcal{Y}, && \leftarrow \text{state, input, output constraints} \\
& x_N \in \mathcal{X}_f. && \leftarrow \text{terminal state constraint}
\end{aligned}$$

The equivalent multi-parametric quadratic program (mp-QP):

$$\begin{aligned}
\mathbf{u}_N^\star(x_0) = \arg\min_{\mathbf{u}_N} \; & \frac{1}{2} \mathbf{u}_N^\top \tilde{Q} \mathbf{u}_N + {x_0}^\top \tilde{H} \mathbf{u}_N \\
\text{s.t.} \; & A\mathbf{u}_N \leq b + Ex_0,
\end{aligned}$$

The Explicit MPC allows to solve the optimization problem off-line:

- The optimal control is an "explicit" function of the state → the on-line operations become simple function evaluations.
- Usually, the control law is a piecewise affine (PWA) function → the controller is stored in a lookup table of affine gains.
- Since both the control law and the cost surface are known (piecewise affine and, respectively, quadratic), stability and performance can be analyzed offline.



(a) critical regions (b) piecewise control action (c) piecewise cost surface

Figure 1: Illustration for the explicit MPC problem (the 2D case)

Objectives

Assuming that we have already computed the list of critical regions and their associated control laws, we are interested in efficient ways to locate the currently active region. This proves surprisingly difficult when there are many critical regions to be searcher (sequential search may be too slow). Thus, we wish to test several methods.

O1) Tree-based approaches **[practică/licență]**

- [10] interprets a PWA function as a weighted power diagram (a generalization of the Voronoi diagram). The Voroi diagram center closest to the current point is identified in a logarithmic time.
- [7] chooses directions and lists all regions which have a non-empty projection to construct a decision tree; choosing good direction is posed as a mixed integer problem in [8]
- [4] combines an orthogonal (axis-orthogonal separation hyperplanes) truncated binary search tree (OTBST) with a lattice representation of the PWA formulation: each node contains a subset of CRs from which the active one is selected by running a lattice formula
- [5] uses a hash table to speed up the point location: a variable-width grid is superposed over the partition, and to which cell is attached a list of CRs
- [6] expands the binary tree description to a multiway m-tree: search operations may be implemented in parallel
- [3] stores the connectivity graph obtained in the construction phase, enumerates the current's CR half-spaces which do not contain x_0 and flips them iteratively to find the new CR
- [2] proposes a hybrid data-structure, composed from a k-d tree (GKDT) which is composed from a k-dimensional tree, hash table and binary search tree (BST) to divide the search tree into multiple sub-trees and traverse them via a hash function on each level.

Theory	Implementation	Tools
●●○○○	●●●○○	Matlab/Python/C++

O2) Lattice-based approaches **[practică/licență]**

[9], for the case of a scalar input, gives a lattice formulation:

- consider the PWA function

$$p(x) = \ell(x, \theta_i) = \begin{bmatrix} x^\top & 1 \end{bmatrix}^\top \theta_i,\ \forall x \in R_i$$

- then there exists the equivalent formulation (disjunctive form)

$$P(x, \Theta, \Psi) = \min_{1 \le i \le M} \left\{ \max_{1 \le j \le M, \Psi_{ij}=1} \ell(x, \theta_j) \right\}$$

where $\Theta = \begin{bmatrix} \theta_1 & \dots & \theta_M \end{bmatrix}^\top$ and $\Psi_{ij} = \begin{cases} 1, & \text{if} \quad \ell(x, \theta_i) \ge \ell(x, \theta_j) \\ 0, & \text{otherwise} \end{cases}$

- retrieving the affine law is now simply a series of min/max operations; various simplifications are discussed
- a similar, conjunctive form (max-min), exists [1]

Theory	Implementation	Tools
●●○○○	●●●●○	Matlab/Python/C++

People

- Florin Stoican
- Ștefan Mihai

Relevant references

[1] J. Xu and Y. Lou. "Error-free approximation of explicit linear MPC through lattice piecewise affine expression". In: (Oct. 1, 2021). URL: https://arxiv.org/abs/2110.00201v2 (visited on 12/14/2021).

[2] X. Xiu and J. Zhang. "Grid kd tree approach for point location in polyhedral data sets–application to explicit MPC". In: *International Journal of Control* 93.4 (2020). Publisher: Taylor & Francis, pp. 872–880.

[3] M. Herceg, S. Mariéthoz, and M. Morari. "Evaluation of piecewise affine control law via graph traversal". In: *2013 European control conference (ECC)*. IEEE, 2013, pp. 3083–3088.

[4] F. Bayat, T. A. Johansen, and A. A. Jalali. "Flexible piecewise function evaluation methods based on truncated binary search trees and lattice representation in explicit MPC". In: *IEEE Transactions on Control Systems Technology* 20.3 (2011). Publisher: IEEE, pp. 632–640.

[5] F. Bayat, T. A. Johansen, and A. A. Jalali. "Using hash tables to manage the time-storage complexity in a point location problem: Application to explicit model predictive control". In: *Automatica* 47.3 (2011). Publisher: Elsevier, pp. 571–577.

[6] M. Mönnigmann and M. Kastsian. "Fast explicit MPC with multiway trees". In: *IFAC Proceedings Volumes*. 18th IFAC World Congress 44.1 (Jan. 1, 2011), pp. 1356–1361. ISSN: 1474-6670. DOI: 10.3182/20110828-6-IT-1002.00686. URL: https://www.sciencedirect.com/science/article/pii/S1474667016437985 (visited on 01/17/2023).

[7] A. N. Fuchs, D. Axehill, and M. Morari. "On the choice of the linear decision functions for point location in polytopic data sets-application to explicit MPC". In: *49th IEEE Conference on Decision and Control (CDC)*. IEEE, 2010, pp. 5283–5288.

[8] A. N. Fuchs, C. N. Jones, and M. Morari. "Optimized decision trees for point location in polytopic data sets-application to explicit MPC". In: *American Control Conference (ACC), 2010*. 00020. IEEE, 2010, pp. 5507–5512. URL: http://ieeexplore.ieee.org/abstract/document/5530979/.

[9] C. Wen, X. Ma, and B. E. Ydstie. "Analytical expression of explicit MPC solution via lattice piecewise-affine function". In: *Automatica* 45.4 (2009). Publisher: Elsevier, pp. 910–917.

[10] C. N. Jones, P. Grieder, and S. V. Raković. "A logarithmic-time solution to the point location problem for parametric linear programming". In: *Automatica* 42.12 (2006). 00060, pp. 2215–2218. URL: http://www.sciencedirect.com/science/article/pii/S0005109806003074.

Robot Operating System 2

Idea

Robot Operating System 2 (ROS 2) is a cutting-edge framework designed for building robot applications, providing tools and libraries to aid in the development and deployment of robotic systems [1, 2, 3]. It emphasizes real-time performance, security, and improved communication.

The framework uses Data Distribution Service (DDS) for its communication layer, offering better support for distributed systems, making it ideal for complex, networked robotic applications. ROS 2 also ensures cross-platform compatibility, enabling developers to work on various operating systems, including Windows, Linux, and macOS, thus broadening its usability in diverse environments.

The capabilities cover essential tools for robot simulation, visualization, and control:

- Gazebo is a powerful simulation tool integrated with ROS 2, allowing users to create accurate and realistic models of robotic systems and their environments. This integration facilitates the testing and development of robots in a virtual space, significantly reducing the need for physical prototypes.
- RViz, another crucial component, is a visualization tool that helps developers to see the data from their robots in real-time. This includes sensor data, robot state information, and the environment, aiding in debugging and fine-tuning robotic operations.
- Furthermore, teleop tools in ROS 2 enable remote operation of robots, allowing developers to control their systems from a distance, which is particularly useful for field robots or those in inaccessible areas.

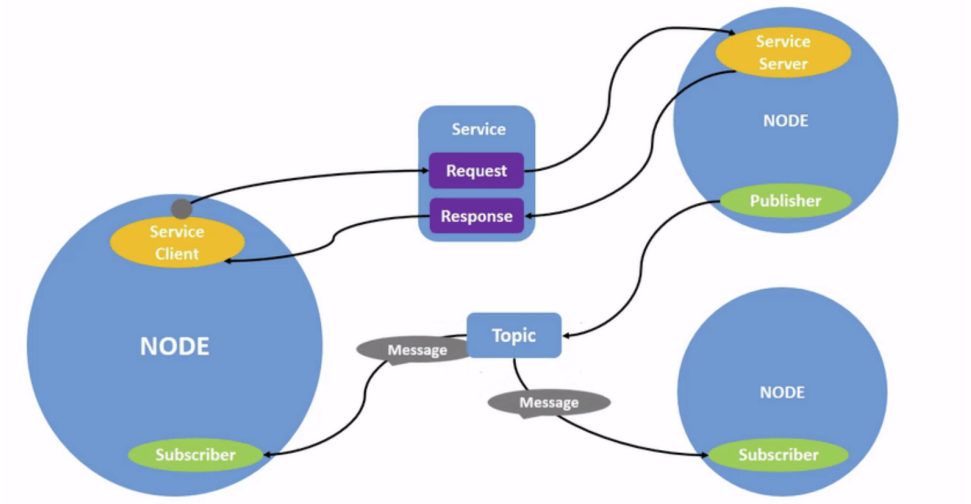


Figure 1: Illustration of ROS2 architecture [1]

Objectives

There are multiple objectives that may be solved together or in parallel:

O1) Design and document a ROS2 implementation **[practică/licență]**

- check existing ROS and ROS2 implementations for useful materials (e.g., RotorS from ETH, The Construct or CrazySwarm)
- construct a "barebones" architecture based on ROS2 and save it as a Docker file

Theory	Implementation	Tools
●○○○○	●●●●○	Python/C++

O2) Design and document worlds for simulation in Gazebo **[practică/licență]**

- gather worlds that can be loaded for simulation from various existing projects
- construct worlds programmatically (Python/Matlab script to add various simple primitive into a predefined space)
- check tools for constructing worlds from Google Maps data (make a virtual landscape/city)

Theory	Implementation	Tools
●○○○○	●●●○○	Python/C++

O3) Design and document models for simulation in Gazebo **[practică/licență]**

- gather models that can be loaded for simulation from various existing projects (TurtleBot, various drones)
- understand and modify the associated URDF files
- add functionalities to standard models (how to add a disturbance, a fault event or uncertain dynamics)

Theory	Implementation	Tools
●○○○○	●●●○○	Python/C++

O4) A lot of effort in Crazyflie and ROS2 (see https://www.bitcraze.io/tag/ros2/), let's check what happens and what we can adapt/use! **[practică/licență]**

Theory	Implementation	Tools
●○○○○	●●●●○	Python/C++

People

- Florin Stoican
- Bogdan Gheorghe
- Daniel Ioan

Relevant references

[1] ROS2. *Tutorials for the Jazzy version of ROS2*. 2024. URL: https://docs.ros.org/en/jazzy/Tutorials.html.

[2] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall. "Robot operating system 2: Design, architecture, and uses in the wild". In: *Science robotics* 7.66 (2022), eabm6074.

[3] F. M. Rico. *A concise introduction to robot programming with ROS2*. Chapman and Hall/CRC, 2022.